Lecture Notes in
Computer Science 1725

Jan Pavelka Gerard Tel
Miroslav Bartošek (Eds.)

SOFSEM'99: Theory and Practice of Informatics

26th Conference on Current Trends
in Theory and Practice of Informatics
Milovy, Czech Republic, November/December 1999
Proceedings

Springer

Lecture Notes in Computer Science 1725
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Jan Pavelka   Gerard Tel
Miroslav Bartošek (Eds.)

SOFSEM'99:
Theory and Practice of Informatics

26th Conference on Current Trends
in Theory and Practice of Informatics
Milovy, Czech Republic, November 27 - December 4, 1999
Proceedings

Springer

Volume Editors

Jan Pavelka
DCIT, s.r.o
J. Martího 2/407, 162 02 Prague 6, Czech Republic
E-mail: pav@dcit.cz

Gerard Tel
University of Utrecht, Department of Computer Science
Padualaan 14, 3584 CH Utrecht, The Netherlands
E-mail: gerard@cs.uu.nl

Miroslav Bartošek
Masaryk University, Institute of Computer Science
Botanická 68a, 602 00 Brno, Czech Republic
E-mail: bartosek@ics.muni.cz

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Theory and practice of informatics: proceedings / SOFSEM '99, 26th Conference on Current Trends in Theory and Practice of Informatics, Milovy, Czech Republic, November 27 - December 4, 1999 / Jan Pavelka . . . (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1999
(Lecture notes in computer science ; Vol. 1725)
ISBN 3-540-66694-X

CR Subject Classification (1998): D, F, H.1-3, C.2, G.2, I.2

ISSN 0302-9743
ISBN 3-540-66694-X Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
SPIN: 10705490 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Preface

This year the SOFSEM conference is coming back to Milovy in Moravia to be held for the 26th time. Although born as a local Czechoslovak event 25 years ago SOFSEM did not miss the opportunity offered in 1989 by the newly found freedom in our part of Europe and has evolved into a full-fledged international conference. For all the changes, however, it has kept its generalist and multidisciplinary character. The tracks of invited talks, ranging from Trends in Theory to Software and Information Engineering, attest to this. Apart from the topics mentioned above, SOFSEM '99 offers invited talks exploring core technologies, talks tracing the path from data to knowledge, and those describing a wide variety of applications.

The rich collection of invited talks presents one traditional facet of SOFSEM: that of a winter school, in which IT researchers and professionals get an opportunity to see more of the large pasture of today's computing than just their favourite grazing corner. To facilitate this purpose the prominent researchers delivering invited talks usually start with a broad overview of the state of the art in a wider area and then gradually focus on their particular subject.

Apart from being a winter school, SOFSEM is a conference where outstanding submitted contributions selected by the programme committee are presented in shorter contributed talks also published in the proceedings. This year the 23 members of the programme committee, coming from 8 countries, evaluated 45 submissions, with several sub-referees joining the effort. After careful review, followed by a comprehensive discussion at a PC meeting held at the end of June, 18 contributions were selected for presentation at SOFSEM '99. At SOFSEM the invited and contributed talks will be supplemented, as usual, by refereed posters and by flash communications.

Throughout its history, SOFSEM has served as a meeting ground for professionals from both the theoretical and the practical side of computing. Alas, the fence dividing those two still exists after all these years. Yes, there have always been holes in it and some of you may insist that they are getting larger lately. Having lived on both sides I find it difficult to judge; at any rate I believe that placing SOFSEM squarely across the fence is and will remain useful.

There are many people whose contribution to the preparation of SOFSEM '99 I acknowledge gratefully – and gladly. Gerard Tel as the co-chair has done much to make the programme committee work smoothly and productively. He has been a great help and a great pal, taking the initiative whenever he sensed (rightly) that it was time to step in. The members of the PC were hard working and disciplined. Some were special: Peter van Emde Boas and Torben Hagerup went well beyond their assignments. The names of all PC members as well as those of the sub-referees are recorded elsewhere. Together they made the selection process – rather difficult this year – balanced and fair. Having a paper refused is not exactly an exhilarating experience, but more often than not the author received a detailed analysis of the contribution, suggesting improvements and providing hints for further research.

It has been proven possible to have a SOFSEM without Jan Staudek leading the organizing committee, without Jiří Sochor serving as the PC secretary, or without Miroslav Bartošek taking care of the proceedings. Nevertheless, I am glad I was lucky enough not to have to try. Working with Jan, Jiří, and Miroslav saved me much sweat and some embarrassment. Now that Springer-Verlag (whose continuing trust in SOFSEM is very much appreciated) publishes LNCS in both printed and electronic form, editing the proceedings is more complicated; fortunately we had Petr Sojka and Aleš Křenek to help, with Petr Sojka taking over most of the burden.

To my colleagues in the SOFSEM steering committee I owe my gratitude not only for guidance and advice, but also for recruiting most of the invited speakers.

During the transition to an international event we have been able to stick to the tradition of making SOFSEM affordable to the public from Central and Eastern Europe and even support the participation of graduate and post-graduate students financially. This is due to the generosity of our sponsors whose names are also recorded in these proceedings.

Having said my thanks I now get to wishes. I recall Jan Staudek responding to complaints about meeting in Milovy again and again by noting that the Hawaii Conference on System Sciences does not move from Hawaii and nobody seems to protest. Well, I must admit that apart from clear air and abundance of vegetation our venue does not offer the sort of attraction Hawaii does. On the other hand, less tourist attraction means less distraction, making the participants huddle together during whatever free time the busy schedule of SOFSEM allows for. This can be used in various productive ways – from planning joint research and projects to sitting at the feet of the veterans and soaking up war stories and tricks of the trade. I hope the participants will enjoy all that SOFSEM '99 has to offer, both on and off its time-table.

August 1999 Jan Pavelka

SOFSEM '99 Committees

Advisory Board

Dines Bjørner	Technical University of Denmark, DK
Manfred Broy	Technical University of Munich, DE
Michal Chytil	ANIMA Praha, Prague, CZ
Peter van Emde Boas	CWI Amsterdam, NL
Georg Gottlob	Technical University of Vienna, AT
Keith G. Jeffery	CLRC RAL, Chilton, Didcot, Oxon, UK
Maria Zemánková	NSF, Washington DC, USA

Steering Committee

Keith G. Jeffery	CLRC RAL, Chilton, Didcot, Oxon, UK
Jan Pavelka	DCIT, Prague, CZ
František Plášil	Charles University, Prague, CZ
Igor Prívara	INFOSTAT, Bratislava, SK
Branislav Rovan	Comenius University, Bratislava, SK
Jan Staudek	Masaryk University, Brno, CZ
Jiří Wiedermann, *chair*	ICS, Academy of Sciences of the CR, Prague, CZ

Programme Committee

Bernadette Charron	LIX Paris, FR
Peter van Emde Boas	University of Amsterdam, NL
Linda van der Gaag	Utrecht University, NL
Torben Hagerup	MPI Saarbrücken, DE
Petr Hanáček	Technical University of Brno, CZ
Václav Hlaváč	Czech Technical University, Prague, CZ
Keith G. Jeffery	CLRC RAL, Chilton, Didcot, Oxon, UK
Shmuel Katz	Technion, IL
Mojmír Křetínský	Masaryk University, Brno, CZ
Klaus-Jorn Lange	University of Tübingen, DE
Bořivoj Melichar	Czech Technical University, Prague, CZ
Jan Pavelka, *chair*	DCIT, Prague, CZ
František Plášil	Charles University, Prague, CZ
Jaroslav Pokorný	Charles University, Prague, CZ
Igor Prívara	INFOSTAT, Bratislava, SK
Branislav Rovan	Comenius University, Bratislava, SK

Peter Ružička	Comenius University, Bratislava, SK
Václav Šebesta	ICS, Academy of Sciences of CR, Prague, CZ
Arno Siebes	CWI Amsterdam, NL
Jiří Sochor, *secretary*	Masaryk University, Brno, CZ
Gerard Tel, *co-chair*	Utrecht University, NL
Philippas Tsigas	Chalmers University, SE
Josef Vašica	DCIT, Prague, CZ
Filip Zavoral	Charles University, Prague, CZ

Sub-referees

Nick Afshartous
Yosi Ben-Asher
Shai Ben-David
Eike Best
Hans Bodlaender
Ivana Černá
Bernard Chazelle
Volker Diekert
Henning Fernau
Ján Hric
Moshe Israeli
Petr Jančar
Václav Jírovský
Michael Kaminski
Antonín Kosík
Lenka Motyčková
Roman Neruda
Jorge Pinto
Hein Röhrig
David Rutter
Ernst Seidel
Michael Sperber
Jitka Stříbrná
Petr Tůma
Marinus Veldhorst
Bram Verweij
Michel Weinfeld
Shmuel Zaks

Organization

SOFSEM '99 is organized by

CSCS, Czech Society for Computer Science,
DCIT, Prague,
Faculty of Informatics, Masaryk University, Brno,
Institute of Computer Science, Academy of Sciences of CR, Prague,
Utrecht University,

in cooperation with

Slovak Society for Computer Science.

Organizing Committee

Jan Staudek, *chair*
Miroslav Bartošek, *vice-chair*
Petr Hanáček
Dana Komárková
Aleš Křenek
Zdeněk Malčík
Tomáš Pitner
Petr Sojka
Tomáš Staudek

Sponsoring Institutions

ApS Brno, s.r.o.
Compaq Computer, s.r.o.
ERCIM, the European Research Consortium for Informatics and Mathematics
Hewlett Packard, s.r.o.
IBM Czech Republic, s.r.o.
Oracle Czech, s.r.o.

Table of Contents

Invited Talks

Trends in Theory

Core Technologies

Software and Information Engineering

From Data to Knowledge

Applications

Contributed Papers

Quantum Challenges*

Jozef Gruska

Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
Email: gruska@fi.muni.cz
URL: http://www.fi.muni.cz/usr/gruska

Abstract. Quantum information processing research has brought deep insights into the quantum potentials of Nature for computing, revealed new information processing primitives, principles, concepts, methods and also brought some spectacular results. On the other side, the problems connected with the design of powerful quantum processors are still seen as immense.

The aim of the talk is to present and to analyse the current situation in quantum information processing and, especially, to point out main quantum challenges that need to be attacked in order to make a further progress in this fascinating and promising area.

1 Prologue

> There are times on which a genius would wish to live. It is not in the still calm of life, or in the response of a pacific situation, that great challenges are formed. ... Great necessities call out great virtues.
>
> Abigoul Adams (1744–1810)

An understanding of the quantum information processing (QIP) fundamentals. principles, laws and limitations, as well as utilization of its potentials, is one of the main challenges of the current science and technology in general, and of the computing and quantum physics in particular.

On one side, there is a strong feeling, based on the historical experiences, and on the recent progress in the area, that the current merging of two arguably most powerful scientific and technological developments of 20th century – quantum physics and computing – will converge into a fruitful marriage with strong impacts on both areas, with a better understanding of the nature of physical and information processing worlds and, perhaps, with a new technology that could represent **the** computer revolution.

On the other side, problems that need to be overcome, in order to make this vision true, are still overhelming, especially on the experimental and technological side, but not only there. In order to overcome these difficulties, if that is

* Paper has been written during the author stay at the University of Nice, Sophia Antipolis, within the PAST program. Support of the GAČR grant No. 201/98/0369 and of the grant CEZ:J07/98:143300001 is also to appreciate.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 1–28, 1999.

at all fully possible, it is useful to identify carefully main quantum (information processing) challenges and to concentrate on dealing with them.

There are many ways how to illustrate the dimension of the potentials the developments in quantum computing can provide and by that also the potential merit of such quantum challenges. Let us start with a general and global view of the history of mankind that distinguishes three basic eras:

Neolithic era. The progress was made on the basis of the development of means to achieve that mankind had enough **food** and whenever needed.

Industrial era. The progress has been made on the basis of the development of means to achieve that mankind has had enough **energy** and whenever needed.

Information era. The progress is being made on the basis of the development of means to achieve that mankind has enough **information** and whenever needed.

The above views point out the key role of information for the future. The next global view concerns the history and the nature of computing.

19th century: Computing was seen as mental processes.
20th century: Computing has been seen as machine processes.
21th century: Computing will be seen as Nature processes.

The last view points out that in order to make a progress in information processing we have to look more into the Nature and to try better to utilise its natural information processing capabilities. The last view points out one direction to go in making use of the information processing potentials of Nature. (Biocomputing seems to be another direction and challenge.)

19th century: Progress was made by the conscious application of the classical mechanics (Newton equations and thermodynamics), to create a basis for the industrial revolution.

20th century: Progress has been made by a conscious application of modern mechanics (Maxwell equations and electromagnetism), to create energy and information distribution networks and tools and to utilize them in order to create a basis for the information revolution.

21th century: Progress is expected to be made by a conscious application of quantum mechanics (Schrödinger equation and information processing), to create a basis for the quantum information processing revolution.

In a complex world we live one often needs simple (but proper) “slogans” to see a right way to go. Slogans presented above suggest that quantum information processing is a big challenge and, naturally, great ideas in this field may have large impacts. The developments during the last years in this area also suggest that in spite of many deep and surprising results and discoveries, the field is at its beginning and “the floor is open for new great ideas to come”.

There are also urgent practical needs to pursue as much as possible the idea of designing quantum computers. Indeed, for a progress in physics and other sciences it would be much useful to be able to use quantum computers to simulate quantum phenomena, what classical computers cannot do (efficiently). In addition, it is becoming clear that in order to be able to make progress in computing technology still for a while according to the Moore law, we need to go for computing power into the quantum level, one way or another.

2 Introduction

> Progress in science is often made by pessimists.
> Progress in technology is mainly made by optimists.

Two outcomes, one on the theoretical level and one on the experimental level, that appeared at about the same time, around 1994, acted as apt killers for quantum information processing – a fascinating challenge that has been developed rapidly since then. Interesting enough, both of these results are of deep interest for security of communication – one of the key problems to deal with in order to put the global information society communication on a secure basis. The first of these results were Shor's quantum polynomial time algorithms for factorization and discrete logarithm computation that would allow, if quantum computers were available, to break many of the currently used systems for encryptions and digital signatures. The second of these results were successful, several kilometers long, transmissions of photons that could be a basis for new ways quantum key generation is performed, representing a significantly new dimension in the security of communication, at which undetectable eavesdropping is impossible, on the basis of the physical laws.

Interesting enough, the basics of quantum mechanics needed to develop quantum information processing have already been known for more than 60 years. From this pont of view it may seem a bit surprising that neither Turing nor von Neumann, one of the fathers of both fast universal computers and modern quantum mechanics, did not try to develop computing principles on the laws of quantum physics, what would better correspond to their times knowledge of the physical world, but they did it on the basis of the classical physics. However, a closer look to the overall state of knowledge and practical needs of their times quite clearly shows why the idea of quantum computing got to be of a broader interest only around 1994–5. At the times of Turing and von Neumann, it did not seem feasible that quantum laws, due to the requirement on the reversibility of quantum evolution, randomness of quantum measurements, uncertainty principle and non-locality, could be utilized to make reversible quantum evolution to perform precise, universal and powerful computations. This started to be seen as possible only after Bennett's result[1] in 1973 – that universal reversible Turing machines exist. Moreover, it was practically impossible to develop a sufficiently justified belief that it could pay off to try to overcome enormous technological problems on the way to the design of powerful quantum computers, that clearly need to be dealt with, until the theory of computational complexity,

[1] Those older and well-known references that cannot be found in the list of references of this paper can be found in [21].

and especially theory of randomized computational complexity, was sufficiently developed, what happened only in the years 1980–1990. At that time physics hardly had available concepts and tools to develop a deeper and broader understandings of the potential quantum computing could provide. Finally, it was just around 1994–5, when sufficient experience in experimental atomic physics, optics and nuclear magnetic resonance has developed to the point to start to consider, at least in a rudimentary form, building of the experimental quantum gates and processors. For example, the idea of the ion trapped technology, that was the first one suggested to experiment in quantum information processing, by Cirac and Zoller, in 1995, was developed only shortly before that. One can therefore conclude that without theoretical advances in the randomized complexity theory and in the experimental atomic physics and optics, it was practically impossible that the idea of quantum information processing would get a significant momentum, no matter how theoretically natural it was already for some years.

Potentials of quantum computers seem to be immense. A quantum computer with “tiny” quantum memory of 200–1000 qubits could be used to break some important current cryptosystems. Even a quantum computer with a “toy-size” memory, of 10–20 qubits, could be used to perform significant quantum experiments to enhance our knowledge of quantum physics and of the other areas of science. However, immense seem to be also problems with designing quantum computers of a significant power. For example, very hard seem to be problems with providing long term quantum memory, with interactions of particles to perform conditional quantum gates, with fighting decoherence and with quantum measurement – to mention some. These problems are so difficult and seem to grow so fast with the number of qubits involved and computation time used, that in spite of the fact that, step by step, 2-, 3-, 5- and even 7-qubit experiments have been already reported [32] recently, the new quantum initiative in Europe, within the 5th Framework program, suggested, in November 1998, to put into the agenda, for the next 4 years, the development of only a 4-qubit quantum processor. In addition, as already mentioned, it is still far from clear whether the design of really powerful quantum computer is fully possible at all. A much better situation seems to be with quantum cryptography, especially in the area of quantum key generation.[2]

Though it is practically impossible to foresee particular developments, as it usually the case in such scientific adventures, one can say quite safely, that in any case we can expect significant increase of our theoretical knowledge of the physical and information processing worlds, their laws and limitations, and that one can expect that at least some byproducts of such research and developments will have important impacts in various areas, as discussed later.

In any case quantum information processing is a fascinating intellectual adventure. Problems range from such esoteric ones as counterfactual computations, quantum teleportation, non-locality, Zeno effect impacts and quantum

[2] S. Braunstein is quoted for saying “We may have sooner quantum telephones than quantum computers”.

entanglement, to such prosaic ones as to how to fight decoherence, how to set up the initial (quasi) pure state and how to perform readouts (measurements).

3 State of the Art

Let us first review briefly some basics of QIP and also some of its main principles, outcomes and obstacles.

3.1 Mathematical Framework

Ideal quantum information processing is performed in Hilbert spaces and they correspond to isolated quantum systems. Real quantum information processing has to be performed in a real world, in a very noisy environment where entanglement with the environment and decoherence dominate. A realistic research in quantum information processing has to take all this into consideration.

Hilbert space is a mathematical framework to describe concepts and processes of isolated quantum systems. It is a complex linear vector space $\mathcal{H}$ on which an *inner product* $\langle . | . \rangle \colon \mathcal{H} \times \mathcal{H} \leftrightarrow \mathbf{C}$ is defined such that for any $\phi, \psi \in \mathcal{H}$ it holds: (1) $\langle \phi | \psi \rangle = \langle \psi | \phi \rangle^*$; (2) $\langle \phi | \phi \rangle \geq 0$; (3) $\langle \phi | \phi \rangle = 0$ iff $\phi = 0$; (4) $\langle \psi | c_1\phi_1 + c_2\phi_2 \rangle = c_1 \langle \psi | \phi_1 \rangle + c_2 \langle \psi | \phi_2 \rangle$. (A quantum interpretation of $|\langle \phi | \psi \rangle|^2$ is as the probability that the state $|\psi\rangle$ evolves into the state $|\phi\rangle$). Using the inner product one can define on $\mathcal{H}$ a norm $||\psi|| = \sqrt{\langle \psi | \psi \rangle}$, a distance $||\phi - \psi||$ and on this basis a topology. An element ψ of $\mathcal{H}$, represented usually by a column vector, is often denoted as $|\psi\rangle$. Notation $\langle \phi |$ is used for a linear functional on $\mathcal{H}$, represented by a row vector, such that $\langle \phi | (|\psi\rangle) = \langle \phi | \psi \rangle$ for any $|\psi\rangle \in \mathcal{H}$.

Elements of $\mathcal{H}$ of the norm 1 are called *pure states* and they correspond to the states of a closed quantum system. Two states $|\phi\rangle$ and $|\psi\rangle$ are called *orthogonal* if $\langle \phi | \psi \rangle = 0$. Physically, only orthogonal states are well distinguishable.

To a (bipartite) quantum system $\mathcal{S}$ that is composed of quantum systems $\mathcal{S}_1$ and $\mathcal{S}_2$ corresponds a Hilbert space $\mathcal{H}$ that is a tensor product $\mathcal{H}_A \otimes \mathcal{H}_B$ of the corresponding Hilbert spaces $\mathcal{H}_A$ and $\mathcal{H}_B$ and its elements are vectors that are tensor products of the vectors of both Hilbert spaces.[3] Elements of a two dimensional Hilbert space $\mathcal{H}_2$ are called *qubits* and they are denoted by $\alpha|0\rangle + \beta|1\rangle$, where $|0\rangle$ and $|1\rangle$ stands for two basis states of $\mathcal{H}_2$. A state of H_{2^n}, an n-fold tensor product of H_2, is called an *n-qubit register* state. Its general form is

$$|\phi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle,$$

where α_x are complex numbers and $\sum_{x \in \{0,1\}^n} |\alpha_x|^2 = 1$.

An evolution of a quantum system (Hilbert space) is represented by a *unitary matrix (operator)* U (such that $UU^* = U^*U = I$, where U^* is a conjugate

[3] The tensor product $u \otimes v$ of n-dimensional vectors $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$ is an n^2-dimensional vector with components $(u_1v_1, \ldots, u_1v_n, u_2v_1, \ldots, u_2v_n, \ldots, u_nv_1, \ldots u_nv_n)$.

transpose of U). A unitary evolution preserves scalar product and represents a reversible process. Unitary operators perform therefore rotations. A quantum evolution is a deterministic process.

An interface between classical and quantum world is made by *quantum measurements.* A (von Neumann or projection) measurement on a Hilbert space is represented by a *Hermitian matrix (observable).* A measurement of a state $|\psi\rangle$ with respect to an observable A in a Hilbert space $\mathcal{H}$ has two outcomes: (1) the state $|\psi\rangle$ is projected into one of the subspaces of $\mathcal{H}$ generated by eigenvectors corresponding to an eigenvalue of A; (2) classical information is produced into which subspace the projection took place. Projection into one of the possible subspaces is done randomly and the corresponding probability is determined by the coefficients at the eigenvectors of that subspace when $|\psi\rangle$ is expressed using an orthonormal basis of the eigenvectors of A. In general, a projection measurement irreversibly destroys a measured quantum state.

Pure states, unitary operators and projection measurements are basic concepts of information processing in idealised closed quantum systems. On a more "real world level" we work with mixed states (density matrices), superoperators (certain positive and trace preserving mappings) and POV (positive operator valued) measurements (POVM).

A *mixed state* $[\psi\rangle$ is a probability distribution on a set of pure states (produced by an imperfect source), notation $[\psi\rangle = \bigoplus_{i=1}^{k}(p_i, \phi_i\rangle$, where $\sum_{i=1}^{k} p_i = 1$. The interpretation is that a source produces the state $|\phi_i\rangle$ with probability p_i. To each such a mixed state $[\psi\rangle$ corresponds a density matrix $\rho_{[\psi\rangle} = \sum_{i=1}^{k} p_i|\phi_i\rangle\langle\phi_i|$. (One way to see how a density matrix $\rho_{[\psi\rangle}$ emerges as a representation of a mixed state $[\psi\rangle$ is that the average value of an observable $\mathcal{O}$ on $[\psi\rangle$, $\langle\mathcal{O}\rangle_{[\psi\rangle} = \sum_{i=1}^{n}\langle\phi_i|\mathcal{O}\phi_i\rangle = Tr(\mathcal{O}\rho_{[\psi\rangle})$, (where $Tr(A)$, trace of A, is the sum of diagonal elements of a matrix A). Such a density matrix captures all and only information that can be obtained by an observer allowed to examine infinitely many times states from the given source. Density matrices corresponding to two different mixed states can be the same.

The so-called *Shannon entropy* of the mixed state $[\psi\rangle$, with density matrix ρ, is defined by

$$QS([\psi\rangle) = QS(\rho) = -Tr(\rho \lg \rho) = -\sum_{i=1}^{n} \lambda_i \lg \lambda_i,$$

where $\{\lambda_i\}_{i=1}^{n}$ is the multiset of eigenvalues of ρ. ($QS(\rho)$ represents the degree of ignorance embedded in the mixed state $[\Psi\rangle$ (density matrix $\rho_{[\Psi\rangle}$)).

Density matrices are used also to deal with a situation that a composed Hilbert space $A \otimes B$ is in a state $|\phi\rangle$. In such a case a density matrix $\rho = Tr_B|\phi\rangle\langle\phi|$, where Tr_B is the tracing out operation over the subspace B, represents all and only information obtained given infinitely many opportunity to examine the subsystem A of the system $A \otimes B$ prepared in the state $|\phi\rangle$. The most general operators that can be applied to density matrices are called *superoperators*, or trace-preserving completely positive mappings. They can be seen as being performed by first making a composition of a given quantum system with an auxiliary system, called *ancilla*, then applying on such a composed system some unitary transformations and, finally, discarding the auxiliary subsystem. Consequently, if a superoperator is applied to a quantum state, then the resulting state can be in a Hilbert space of larger dimension than that of the input state. Examples of superoperators are quantum encoders, channels and decoders.

POV measurements are the most general type of quantum measurement. They can be seen as follows: one first compose an ancilla with the given quantum system, then performs unitary transformation on the composed quantum system and, finally, performs projection measurement on the resulting state of the ancilla. (Technically, any such a measurement in a d-dimensional Hilbert space is given by a collection of semi-definite Hermitian matrices $\{E_i\}_{i=1}^k$ such that $\sum_{i=1}^k E_i = 1$. An important point is that k can be arbitrarily larger than d.) If, using such a measurement, a system in state ρ is measured, then the probability of obtaining the ith result is $Tr(\rho E_i)$. For details see [21].

3.2 Quantum Resources

It has been demonstrated that by using quantum algorithms, networks and communication protocols one can obtained a significant (up to exponential) increase in performance: computations with less quantum operations and communications with less bit (or qubit) exchanges. The power of quantum communication and computation is the consequence of three quantum phenomena:

1. **Quantum superposition**. An n qubit quantum register can be simultaneously in a superposition of 2^n basis states $\{|x\rangle \mid x \in \{0,1\}^n\}$. In addition, an equally weighted superposition of all its basis states of H_{2^n},

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle,$$

an important initial state of many computations, can be created in a single step, from a simple basis state $|0 \ldots 0\rangle$, using n simple one-qubit (Hadamard) operations $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.
On the other side, a superposition of 2^n basis states can evolve in one step into a superposition consisting of only one basis state. If one then makes the measurement of the resulting state, with respect to the given basis, then the outcome is obtained with probability 1. This is often used to get useful classical information from a complex quantum evolution because most of the quantum information contained in a quantum state is inaccessible.

2. **Quantum parallelism**. In one step of a quantum evolution exponentially many classical computations can be “performed”. For example, for any function $f\colon \{0,1,\ldots,2^n-1\} \to \{0,1,\ldots,2^n-1\}$, there is a unitary mapping $U_f\colon |x,0\rangle \to |x,f(x)\rangle$ and if U_f is applied to an exponentially large superposition

$$\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i,0\rangle,$$

then, in one step, we obtain the state

$$\frac{1}{\sqrt{2^n}}\sum_{i=0}^{2^n-1}|i,f(i)\rangle, \tag{1}$$

and therefore exponentially many values of f can be computed in one step.

3. **Quantum entanglement**. Some quantum states of a bipartited quantum system cannot be expressed as a tensor product of the states of its subsystems. For example, the state $\frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$, called singleton. Such pure states are called *entangled.* Entangled states are a puzzling phenomenon because they exhibit non-local features. Indeed, it may be the case that two particles in the singleton state are (very) far from each other. In spite of that any measurement of one of the particles immediately determines the state of the second particle.[4]

   The concept of entanglement, or inseparability, is defined also for density matrices. A density matrix ρ of a bipartite quantum system $A\otimes B$ is inseparable, if it cannot be written in the form $\rho=\sum_i p_i\rho_i^A\otimes\rho_i^B$, where ρ_i^A (ρ_i^B) is a density matrix of the subsystem A (B).

   Quantum entanglement is considered as a precious quantum resource due to which quantum computation and communications can outperform classical ones. At the same time quantum entanglement is an important demonstration of non-local features of quantum mechanics that has so much puzzled already generations of scientists. On the other side, it is to a large extent due to the existence of the entangled states why quantum computers are so hard to build!

   A study of methods to create, purify, distribute and consume quantum entanglement belongs to the most important current research subjects in the area of quantum information processing.

4. **Quantum measurements.** Surprisingly, quantum meusurement should also be seen as a powerful computational primitive. Indeed, by performing one measurement step one can achieve, in some cases, that a solution is found of a complicated system of constrains and this solution is incorporated into the resulting projection (state). For example, by measuring the state (1), with respect to the f-register, a solution of the system of constrains

$$\bar{f}\in\{f(i)\,|\,0\le i<2^n\},\quad I=\{i\,|\,f(i)=\bar{f}\}$$

   is found and “included” into the resulting state $\frac{1}{|I|}\sum_{i\in I}|i,\bar{f}\rangle$.

[4] Quantum entanglement can be seen as representing inherently quantum form of information distinguished from the classical one also in the following sense: classical information can be copied, but can be transferred only forward in time. Quantum information cannot be copied, but can be used to connect *any* two points of space and time (and can therefore be seen as propagating also backwards in time, as in the case of quantum teleportation).

3.3 Quantum Principles

Several quantum principles play an important role in quantum information processing and communication.

1. Quantum information cannot be (perfectly) cloned.[5]
2. Gaining classical information from quantum states causes, in general, their disturbance.
3. Quantum measurements are, in general, irreversible. (However, as discussed later, in some interesting cases this does not have to be so.)
4. Nonorthogonal quantum pure states cannot be faithfully distinguished.[6,7]
5. Mixed states with the same density matrix cannot be physically distinguished.
6. Entanglement cannot be increased by local unitary operations and classical communication.

All these principles influence quantum information processing in several ways. For example, they make some possible (impossible) classical tasks to be impossible (possible) at the quantum level.

3.4 Apt Killers

Several results have been obtained in quantum information processing that have turned attention of the whole science and technology community, and also of many outside of this community, to quantum computing and that have contributed much to the excitement this field creates and to the support it gets.

1. Shor's algorithms for factorization and discrete logarithm computation. These results implied that potential quantum computers, even if designed in few decades, could jeopardize security of some current cryptosystems and digital signature schemes.[8]

[5] The so-called no-cloning theorem says that there is no unitary transformation U such that for any qubit state $|\phi\rangle$, $U(|\phi\rangle|0\rangle) = |\phi\rangle|\phi\rangle$. (An amazing result – ships can be cloned, but photons not.)

[6] The need to distinguish nonorthogonal states occurs in many applications and quantum algorithms.

[7] Undistinguishability of nonorthogonal states is also a positive phenomenon. For example, compression of a sequence of nonorthogonal quantum states can go beyond the limits for the compression of classical bits, as stated by the Shannon theorem. Moreover, quantum cryptography is much based on the fact that if a quantum system is prepared in one of the nonorthogonal states, then any attempt to distinguish these two possibilities leads necessarily to a disturbance.

[8] These results were followed soon by a more general one, almost unnoticed by the community at large, due to Boneh and Lipton [5] showing that with quantum computers one could break also other cryptosystems, including now so popular elliptic curve cryptosystem based on the elliptic curve discrete logarithm.

2. The existence of quantum error correcting codes and of a set of universal quantum fault tolerant gates. These results made the vision of quantum computer to be far more a reality than before.
3. Quantum cryptography – theoretical and experimental. More exactly, quantum methods of unconditionally secure key distribution that very soon followed by successful experiments.

3.5 Obstacles

Decoherence remains the problem number one, in spite of a variety contributions in the area of quantum error correcting codes, fault tolerant computation, concatenated codes, and quantum repeaters that show that reliable memory, transmission and processing of quantum information, in time and space, is, in principle, possible.

In spite of that there are doubts when and whether we can really have powerful quantum processors. The research in quantum information processing can be seen as a constructive "fight" between pessimists and optimists, with optimists taking currently an edge.

4 Quantum Challenges

Let us now turn to the main topic of this paper, to major quantum challenges in the area of quantum information processing.

4.1 Quantum Processors

Progress and success in designing quantum processors is surely of a very large (almost of the key) importance for the overall standing of the whole field of QIP. The main burden here is on (experimental) physicists and engineers, but an involvement of (theoretical) computer scientists can also bring large benefit. By creating and investigating models, computation modes and methods of suggested and developed technologies, there is a possibility to contribute to and to speed up their development and utilization. The case of NMR technology and computations, and theoretical contributions to them, is a good example in this direction. Two main challenges in this are can be seen as follows.

1. The search for a proper technology for QIP, suitable to implement easily and reliably a universal set of quantum computational primitives, setting of the initial state, readouts, and to manage decoherence.
   Three technologies have been especially intensively developed and investigated so far: nuclear magnetic resonance (NMR), trapped ions (especially a chain of ions in a linear radio-frequency trap), and cavity QED. Especially NMR technology has been studied and developed much and used to make 2- to 7-qubit experiments [31]. However, none of these technologies seem to scale well and one can expect that in order to have a breakthrough in this

area it is necessary to turn to silicon based technologies as quantum dots [17] or to Kane's [29] proposal. (The basic idea is that spins of electrons confined to localized states in solid-state structures have the prospect of serving as qubits. Various physical mechanisms for implementation of quantum gates using such technologies have been proposed and are under investigations.) Another possibility, that needs to be explored, is to use superconductors and Josephson junctions.

2. The search for simple universal sets of quantum computational primitives is another fundamental and important task. The results in this area can much influence the choice of suitable technologies and also the search for such technologies.
   Progress in this area has been remarkable and it is worth to summarize: (1) Deutsch's three-qubit gate (1989); (2) DiVincenzo's two-qubit universal gate (1995); (3) Barenco et al. [2] universality of the XOR gate and one-qubit gates; (4) universality of the XOR, Hadamard and $\begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i}{4}\pi} \end{pmatrix}$ gates, due to [6] (5) GHZ state $\left(\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)\right)$, Bell measurements and one-qubit quantum gates [20].

Two other challenges in this area are:

1. Experiments – proof-of-the-principle demonstrations of basic quantum computational primitives (entangled states (especially of more qubits)), teleportation, XOR gates, Bell measurements, Zeno effect ...) and experimental verification of simple quantum algorithms, protocols, error correcting codes, and fault-tolerant gates.
   Insbruck's experimental demonstration of quantum teleportation has been considered as one of the three most important experimental achievements in physics in 1998. Various other demonstrations of quantum teleportation has been reported. However, as analysed by Vaidman [46], none of the experiments till then could be actually seen as a 100% demonstration of quantum teleportation. Something was always missing (for example Bell measurements could not be performed fully reliably). In spite of that Vaidman [46] concludes that though a perfect teleportation of unknown quantum state has not been achieved, the experiments performed so far show that it can be done. In addition, the so-called teleportation of continuous variables, suggested by Braunstein and Kimble [8], has been fully demonstrated by Furusawa et al. [18], being the first reliable teleportation experiment.
   Much more insight is needed into quantum teleportation and quite a bit of work has been done in this area. For example, Pati [41] have shown that 1 bit is sufficient to "teleport" an *known* quantum state using an EPR-channels and 1.19 bits are needed in average in the case Alice and Bob share local hidden variables. Hardy [23] challenges a widespread belief that teleportation is a non-local effect by constructing a local theory in which cloning is not possible, but teleportation is. Deutsch and Hayden [16] also see quantum teleportation as a local phenomenon.)

2. Theoretical studies of models and computational modes corresponding to particular technologies. For example of ways to establish an initial state, to perform measurements (readouts), and so on.

4.2 Quantum Cryptography

The main goal in this area is to come from the current experimental stage to a development stage and to practically useful quantum cryptographic systems. However, there may still be a long way to do so.

If judging solely by the distance for which “successful” photons transmissions experiments have been performed, then the progress has been remarkable: from 32 cm in 1989 to the recent 47 km using optical fibres and 1 km (0.5 km) in open air at night (day) time – with 2 to 7 km experiments in open air and day time transmissions announced – see [21] for references. However, a closer look at other quantitative and qualitative characteristics of the experiments performed, reveal that there is still very much to improve to get a quality needed for getting to the development and application stages.

Several major challenges in this area can be identified.

1. Earth-to orbit experimental transmission of photons or other particles. (This would culminate the current proof-of-principle experimental stage in quantum cryptography.)
2. To study limitations of the real quantum transmissions and to explore their quantitative and qualitative characterists (speed, error rate, ...).
3. To explore theoretically and experimentally tools that allow transmission of quantum information for long distances and/or long time periods. For example, quantum repeaters, concatenated codes and so on.

Several other challenges deal with the unconditional security of quantum cryptographic protocols – actually the key problem of quantum cryptography.

4. An identification and a study of the main cryptographic attacks. It is the existence of sophisticated quantum attacks what makes the proof of unconditional security of quantum key generation protocols so difficult. In order to see the dimension of the problem let us list main types of quantum attacks an eavesdropper can perform.
   (a) *Intercept-resend attacks.* The eavesdropper measures each signal sent and then resends it.
   (b) *Individual particles attacks.* The eavesdropper tries to learn as much as possible from each signal sent by first contacting the signal with an ancilla and then performing a measurement on ancilla.
   (c) *Collective attacks.* The eavesdropper brings all transmitted particles into interaction with an ancilla and at the end measures all ancillas together to extract some information about the key.
   (d) *Coherent or joint attacks.* Instead of measuring the particles while they are in transit, the eavesdropper regards all transmitted particles as a single entity. She then couples this entity with an ancilla. Afterwards she

sends particles and keeps the ancilla. After the public interactions the eavesdropper extracts from ancilla some information about the key.

Other types of attacks have been identified in the experimental cryptography:

(a) *Trojan horse attack.* This refers to the possibility that an entanglement with environment can "open the door" and let a "Trojan horse to get in and to gather information about quantum communications in the cryptographic protocols".

(b) *Beam splitter attacks and photon number splitter attacks* (Bennett et al., 1992, N. Lütkenhaus). This refers to the fact that it seems to be beyond current technology to produce perfect single photon pulses. An eavesdropper can therefore test whether there are more photons in a pulse and if this is the case then, using a beam-splitter, get one of them. Under some situations this can be utilise to get full information about the polarization of transmitted photons and about the key.

5. Security of quantum key generation (QKG) protocols. Mayers and Yao [38] have shown unconditional security of the protocol BB84 under the assumption that the photon source is perfect. The proof is of the computational complexity type and quite complex and so are final technical statements. In addition, the proof does not provide a method how to calculate the error rate for real implementations. Much simpler is the recent proof by Lo [35] – see also the web updates to[21]. However, for a more sophisticated entanglement-based protocol in which the key generating parties need quantum computers. A search for new protocols and proofs of their unconditional security will surely continue.
6. Security of cryptographic protocols. A big disappointment was a discovery by Lo and Chau, 1996 and Mayers in 1997, that unconditionally secure bit commitment protocols – an important primitive for quantum cryptographic protocols – do not exist. The result implies impossibility of unconditional security for the ideal quantum coin-tossing protocols, and for 1-out-of-2 quantum oblivious protocols. Unconditional security was shown only for a very simple *quantum gambling* protocol [19]. An important open problem is whether there is an unconditional secure cryptographic protocol for *unideal quantum coin-cossing problem* (see the web update to [21] for more details).

4.3 Quantum Entanglement

Almost all deeply quantum problems in QIP deal, in some way, with quantum entanglement that is considered to be the most precious quantum resource. A better understanding of the entanglement, of ways it is created, stored, manipulated (transformed from one form to another), transmitted (even teleported) and consumed (to do useful work), as well as of various types and measures of entanglement, are theoretically perhaps the most basic tasks of the current QIP research, which could also be characterized as being in the stage of gathering phenomenology. Let us summarize main challenges in this area.

1. *How to determine (in a sufficiently easy way) whether a given state is entangled.* For pure states this is easy, but for density matrices the problem is not solved yet. A well known is simple Peres' [42] necessary *partial transpose condition* which has been shown to be also a sufficient condition, by Horodecki et al. [28] for the case of 2×2 and 2×3 compound systems. To solve the problem in a general case is a hot current research topic.
2. *Measures of entanglement.* Some quantum states, say $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, are clearly more (maximally) entangled, than others, say $\frac{1}{\sqrt{k}}|00\rangle + \sqrt{1-\frac{1}{k}}|11\rangle$, $k \gg 2$. The amount of entanglement in a maximally entangled state of two qubits is said to be one *ebit.*[9] For pure quantum bipartite states $|\psi_\alpha\rangle = \alpha|00\rangle + \sqrt{1-\alpha^2}|11\rangle$, the *entropy of the entanglement*

   $$E(|\psi_\alpha\rangle) = -\alpha^2 \lg \alpha^2 - (1-\alpha^2)\lg(1-\alpha^2),$$

   is a good measure of entanglement.[10] For mixed states (density matrices) it seems that there is no single measure of entanglement that is the best.
   Several measures of entanglement of bipartite states have been introduced and investigated (as well as relations among them) – see [21] for a overview. For example: (a) *entanglement of formation*, $E_f(\rho)$, the least expected entanglement of any ensemble of pure states with ρ as its density matrix; (b) *entanglement of distillation*, $E_d(\rho)$, is the maximum yield of singleton states that can be produced from the given mixed state by local operations and classical communication; (3) *total entanglement*, $E_F(\rho) = \lim_{n\rightarrow\infty} \frac{E_f(\rho^{\oplus n})}{n}$, called also (a regularized version of the) "entanglement of formation". ByHorodecki et al. [26] $E_d(\rho) \leq E(\rho) \leq E_F(\rho)$ for any "good" measure of entanglement..
3. *Creation of pure (maximally) entangled states.* One can consider either a physical source to produce a (mixed) entangled state, or ways to produce entangled states form unentangled ones (i.e. $\mathrm{XOR}\Big(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle\Big) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, or procedures that transform noisy entangled states into pure entangled states (entanglement purification, or distillation) as well as procedures that produce maximally entangled states from weakly entangled states (entanglement concentration). All these cases have been investigated, but much more research is needed to get efficient techniques. In some important cases, say for quantum teleportation, or quantum cryptography, maximally entangled states are needed. Entanglement purification techniques belong to the most important ones in the area of entanglement manipulation.
4. *Operations with entangled states.* Two types of operations are of interests: (1) operations that do not change the amount of entanglement and only change it from one form to another; (2) operations that change the amount

[9] It is also said that two particles in a maximally entangled state form an ebit

[10] Entropy of the entanglement provides also an elegant characterization of the potential to create n singletons from k states $|\psi_\alpha\rangle$ using only local operations. Indeed, as shown by Bennett, Bernstein, Popescu and Schumacher in 1996, $E(|\psi_\alpha\rangle) = \lim_{n,k\rightarrow\infty} \frac{n}{k}$.

of entanglement. From the operations of the second type let us mention entanglement splitting [9] – how and when two parties sharing an entangled state provide a third party with some of their entanglement – and related distributed entanglement, see [15] entanglement swapping and so on.

Of a special interest has been the problem in which case one can transform one entangled state, say $|\phi\rangle$, into another one, say $|\psi\rangle$, by local operations and classical communication. The problem has been solved by Nielsen [40] who found a simple necessary and sufficient condition.

5. *Types of entanglement and the structure of the space of entangled states.* The research in this area seems to be at the very beginning and one can expect a lot to come. An important and surprising discovery along these lines has been made by Horodecki et al. [28]. They should that there exists a *free*, distilable, entanglement, and a *bound*, nondistilable, entanglement.
6. *Entanglement as a computational and communicational resource.* Such a role of entanglement is often much emphasized. However, much more research seems to be needed to get a better understanding what such a general thesis really means. For example, Lo and Popescu [37] explored a question whether one needs classical communication for such entanglement manipulations as entanglement concentration (a transformation of arbitrary states into singlets) and entanglement dilution (transformation of n singletons into a given bipartite state) and showed that these operations practically do not need classical communication (for concentration) or the amount of classical communication converges to 0 with n going to infinity (for dilution).
7. *Entanglement as a computation and communication primitive.* Gottesman and Chuang [20] have shown that by teleporting qubits through special entangled states one can create a variety of interesting gates and that allowed to establish that the GHZ-state, quantum measurement and one-qubit gates form a universal set of quantum computation primitives.
8. *Simulation of entanglement by classical communication.* If the measurements of two particles in an entangled state are space-separated, then, of course, there is no way to simulate them classically. However, if measurement events are time-separated, then such a simulation is possible andBrassard et al. [7] establish a cost of such a simulation for the case of the system in n Bell states.
9. *Multi-particle entanglement.* Understanding, characterization, classification and quantification, as well as manipulation of multi-particle entangled states, is a very important and difficult problem. Perhaps the main current challenge of quantum information theory, that needs to be explored in order to get more insights into the possibilities of distributed quantum computing and networks. The problem is also of interest outside of QIP, as discussed in Section 4.13.

   Error correcting codes are an important example of an area where multi-particle entanglement plays a crucial role and where quite a few insights have been obtained about it.

10. *Analogies and principles.* For the overall development in the field of QIP it is also of importance to try to discover analogies and general principles that are valid but cannot be derived directly from the current quantum mechanics. As an example, let us mention attempts to develop an analogy between the role information plays in physics and that energy plays in thermodynamics, as well as such principles as *no-increasing of entanglement* principle which says that entanglement of a compound quantum system does not increase under unitary processes in one of the subsystems [27].

4.4 Decoherence

The understanding and managing of decoherence is of crucial importance for experimental QIP and for the design of quantum processors. However, decoherence is also a crucial theoretical problem.

The view of decoherence still much differ. From more pragmatic ones to highly theoretical ones. For example, that “decoherence is connected with the storage of information about the decohering system somewhere in the universe” and “decoherence is a process of coupling of a quantum system with its environment” or “decoherence can be regarded as measurements of the quantum system by the environment”.

In spite of an intensive research in this area there seems to be visible only a moderate progress. To find ways to cope with decoherence is often considered as the task for physicists and experimental QIP. However, the field seems to need brand new ideas and it seems that it is one of the areas (theoretical) computer scientists should enter.

Let us identified some subareas on which a larger attention of researcher should focus.

1. An understanding of the decoherence. A realistic investigation of the sources of decoherence and of the forms decoherence exhibits itself, as well as of impacts it can cause.
2. The development of tools to fight decoherence. Two main types of such tools have been identify: (a) active, for example error correcting codes – they fight decoherence by repeated applications of error correction procedures; (b) passive – for example decoherence free subspaces (DFS) – in this case the structure of the particular decoherence processes is used to protect coherence and the symmetries in the decoherence process are utilized. (DFS seem to be ideal for quantum memory applications, but for quantum computing applications they should be combined with QECC (even if DFS can be seen as a special type of QECC with very simple recovery operators [1].
3. Making a positive use of decoherence. It becomes a wisdom in QIP that everything bad is good for something. Let us therefore try to make a positive use of decoherence. By Knight et al. [30] decay can lead to entanglement of quantum systems and detection of decay can be used as a method of state preparation (and can also allow a wide range of communicational tasks – even entanglement).

4.5 Managing Decoherence and Imperfections

One area where unquestionably important progress has been made, with potentially broad impacts also outside of the quantum information processing applications, and where new ideas and large progress are expected, and much needed, concerns the problem how to manage imperfections of quantum processes due to the decoherence, decay, noise and imprecisions – how to protect quantum states from interactions with the environment.

Imperfections of quantum processes and their coupling with environment, that make quantum coherence so fragile, are the main argument of quantum pessimists against quantum computing and main worries of quantum computer design optimists. Impossibility to use the majority voting techniques, so useful in the classical case, due to the no-cloning theorem, and other special properties of quantum states and processes, created a belief that error correcting codes, so useful in the classical case, are impossible in the quantum case.

Shor, in 1995, and a bit later also Steane, found ways how to use quantum entanglement to design quantum error correcting codes (QECC). (QECC also allow to keep unchanged un unknown quantum state in a noisy environment – i.e. to create a long-term quantum memory.) Shortly after, by Shor, quantum fault-tolerant computation was shown to be potentially possible. (Its existence is an important fact because the main real problem with error correction are potential new errors introduced by the error correction procedures themselves.)

In order to deal with imperfections at quantum processes the following techniques have been invented and are expected to be further developed.

1. Quantum error correcting codes. In order to be able to correct t (independent) errors on *real qubits,* the basic idea is to encoded the basis states, $|0\rangle$ and $|1\rangle$, by entangled states (called *logical qubits*) of n qubits such that no information resides in any subset of $2t$ qubits, i.e. the density matrices of any $2t$ qubits are random. Therefore information can be encodded using n qubits, but encoded information has a "global character". There is no way to access any information just by measuring only few qubits.

   A variety of quantum error correcting codes and methods to design codes have already been developed (see [21], for a presentation and references). Progress in this area has been due to the following factors: (1) discretization of errors has been shown to be possible; (2) error correction principles have been discovered; (3) methods to design codes have been found.

   The area seems to be getting firmly into the hands of error-correcting specialists. Progress is expected to come from the following directions: (1) deep techniques of the classical error correcting codes will be adopted to the quantum case; (2) QECC will be developed for specific quantum processes, with specific types and patterns of errors, making use of specific physical properties of these processes; (3) QECC will be developed also for the case of collective errors and for quantum systems that are not decomposable into qubits [32].

2. Development and exploration of other methods of fighting decoherence as entanglement purification, symmetric subspaces, decoherence-free subspaces, making use of the quantum Zeno effect (QZE) and so on.[11]
3. Development of quantum error prevention and detection techniques.
4. Quantum fault-tolerant computations and gates. Two basic approaches are:
   (a) An algorithmic approach. To design methods to perform fault-tolerant computations on logical qubits to simulate, in a quantum fault-tolerant way, quantum gates. Shor showed in 1996, that quantum fault-tolerant computation is possible and showed a way to make fault-tolerant implementation of a set of universal gates. (His results are theoretically important and encouraging, but suggested constructions are two complex for applications.) Since then significant progress in this area has been made (and is still needed). See, for example, much simpler set of universal fault-tolerant gates, due to Boykin et al. [6] and a method to design fault-tolerant gates due to Gottesman and Chuang [20].
   (b) A hardware approach. To develop physically inherently fault-tolerant gates. Kitaev has made such a proposal in 1997. However, the point is to come with an idea implementable with the current technology.

4.6 Quantum Measurement

This is, perhaps, after quantum non-locality, the most controversial problem of quantum mechanics, studied and discussed for more than 60 years, without a really breaking success.

The problem of measurement has already been studied from many points of view, even on the level of the philosophy of sciences. Experimental physicist did not really run into difficulties with the Copenhagen interpretation of quantum measurement that sees the projection measurement as an external intervention into quantum evolution, but leaves open some very basic questions how measurements are performed in Nature and how much resources they need. Quantum information processing needs require to deal with this problem from new points of view and with new aims, that may be beneficial for a better understanding of quantum measurement. Let us summarize some of the new questions.

- How much resources quantum measurements really need? This question has been left practically intact till now. Actually, there has not been so much a real need to answer it. However, now, when one searches for quantitative statements concerning the power and limitations of quantum information processing, this question should not be left open. The answer to this problem can be of importance for the overall view of the potentials of QIP. In principle, it could happen that resources required grow up exponentially with the number of qubits needed. This may have significant impact on quantum computational complexity aims, methods and results.

[11] QZE is the name for the phenomenon of freezing (slowing down) the quantum evolution of a frequently (continuously) measured quantum system.

- How important computational primitive quantum measurements actually are? As observed by Castagnoli et al. [12], by performing a measurement one can get, in one computational step and without apparently “any resources”, incorporated into the resulting projection, a solution of a complicated system of constrains (inequalities). A projection measurement can therefore be a powerful computational primitive and this is actually essentially made used in some efficient quantum algorithms. With this in mind the question of the amount of the resources quantum measurements need is getting a new dimension and urgency.
- Which quantum observables are (currently) implementable and what kind of resources they need. Quantum theory allows us to use a rich variety of observables. Far less is known about a proper use of them. In addition, it is not clear which ones are (currently) implementable, and with which technology. For example, there are still hardly good ways known to distinguish faithfully four Bell states by one observable.

Remark. As pointed out by Reif [45], quantum computing research has not been paying attention to one very important issue. Namely, to the overall amount of such resources as energy and volume quantum computation requires. The problem is that, at least hypothetically, the need for these resources could grow up so fast (exponentially) with the number of qubits involved, that this could discard all advantages quantum superpositions, parallelism and entanglement provide. The key problem here are resources measurements can need.

Since quantum evolution is unitary, and consequently reversible, we can assume that the amount of energy it needs is negligible. On the other side, the problem how much energy and volume quantum measurements need is open and seems to be very hard.

There are several fundamental issues involved. The nature of quantum measurement and the amount and types of quantum measurements one really needs.

On the foundational side, this is an opportunity to realize importance of the fundamental difference between two basic approaches to quantum measurement.

The Copenhagen interpretation that assumes that a quantum measurement is a basic operation which is assumed to be performed by a macroscopic device (of the classical world),

The von Neumann interpretation that considers the measured object (state) and also the measuring device as parts of a larger quantum system. A measuring device and the measurement process themselves are then seen as being done by microscopic systems and subjected to quantum effects.

For experimental physics, where quantum measurements are performed by macroscopic devices, the von Neumann interpretation of quantum measurements has not been really relevant.

A different story may be in the case of quantum information processing in the case of larger amount of qubits. In this case the von Neumann interpretation can be both of interest and importance for at least two reasons.

1. As pointed out by Hay and Peres [24], predictions provided by these two two interpretations can be different
2. It is possible that the volume for quantum observation/measurement grows very fast with the number of qubits involved.

In any case, (good) bounds are needed for total energy consumption and total volume for quantum measuring devices and measurements.

Reif [45] provides arguments, but not a proof, that even attempts to do only "approximate quantum measurements", that may be sufficient for quantum computations, may require exponential resources.

A related hard problem is that of the amount of measurement operations we really need for various computational tasks. For some models, say one-way quantum finite automata, there is an essential difference concerning the recognition power between the models requiring measurements after each operation and the model with only one measurement, at the end of each computation.

As pointed out be Bernstein in 1997, in the case of quantum Turing machines, one can postpone measurements to the end of computation, using a special XORing trick. However, the problem needs to be more investigated.

It is also natural to ask an extreme question whether we need measurements at all. Whether and when it is sufficient to consider as the output the final quantum state. For a more detailed discussion on this subject see [45].

4.7 Quantum Information Theory

There is a belief that quantum information theory could develop to a theory that would be used to explain and interpret all quantum phenomena. The challenge is, if possible, to do so.

In a broad sense, such areas as QECC and entanglement are parts of quantum information theory. From the other major areas let us mention:

1. An abstraction and study of the main quantum information processing primitives and relations among them.
2. The development of the basic concepts of quantum information theory that would parallel and generalize those in the classical information theory.
3. The development of a quantum analogue of the *algorithmic information theory* of Chaitin. The basic step here is to find quantum analogues to such concepts as Chaitin (or self-delimiting Kolmogorov) complexity. The first more involved attempt to do so has been due to Vitanyi [47]. He has introduced two variants of quantum Chaitin complexity of pure quantum states. One is through the length of the shortest classical program generating or approximating a given pure state. The second approach is to consider the length of the shortest quantum qubit program generating or approximating a given pure quantum state. Quantum laws, especially no-cloning theorem, make the development of the corresponding theory not easy. Additional complications come from the fact that the number of possible quantum Turing

machines as well as of quantum states is uncountable. A fixed universal quantum Turing machines cannot therefore generate all pure states – some have to be approximated.

4. An understanding of quantum channels and their capacity to transfer classical and quantum information [3]. In order to communicate through quantum channels three communication primitives can be used: bit, qubit and entangled states (ebits). A variety of the concepts of quantum channel capacities have been identified and investigated. In the case that quantum states are transmitted through a quantum channel and in order to do that classical communication between the sender and the receiver in both ways is possible, then we speak about the *assisted quantum capacity*, Q_2. In the case quantum states are transmitted, but no classical communication is allowed, we speak about the *quantum capacity*, Q. Finally, in the case that classical information is transmitted, we speak about *classical capacity*, C. In the last case there are still four special subcases to consider, depending on whether encoders and decoders are classical or quantum.

   A deeper study of these capacities in general and for particular quantum channels has still to be done. One of the open problems is whether $Q_2 > C$ for some channels.

   The quality of quantum transmissions is measured by the *fidelity* – a similarity between input and output. In the case a pure (mixed) state $|\psi\rangle$ (ρ) is transmitted with probability p_i to the mixed state w_i, the fidelity of the transmission is defined by $\sum_{i=1}^{k} p_i \langle\psi_i|w_i|\psi_i\rangle$ $(\sum_{i=1}^{k} p_i(Tr\sqrt{\sqrt{\rho_i}w_i\sqrt{\rho_i}})^2)$.
5. The study of the data compression methods and limitations. Schumacher's quantum data compression theorem showed achievable theoretical limits for quantum data compression. The problem of universal, efficient and simple compression methods has been solved to some degree. The subject of quantum data compression is, however, of such importance that much more research into the subject is needed.[12]

4.8 Bypassing Quantum Limitations

Physical limitations cannot be really fully bypassed, but there is a strong need to explore them more carefully from the information processing point of view. What they really say, what is the whole truth about them and how important they really are, especially for quantum information processing.

Let us discuss from this point of view some of the limitations.

1. No-cloning limitation. Perfect cloning is impossible. However, how about an imperfect one? Imperfect copies can be perhaps useful in some cases. How many and how good copies we can make using some (universal) gates?

[12] Observe that unknown quantum state cannot be copied and therefore a compression of unknown quantum states has to be made "blindly", without learning these states. It is remarkable that quantum compression of known quantum states cannot be done more efficiently than that of unknown states, due to Schumacher's theorem.

What kind of properties they have? Bužek et al. [10] have investigated such problems.
How much is cloning actually needed and in which cases it can be bypassed? For example, there was quite a strong belief till 1995 that quantum error correcting codes are impossible and no-cloning theorem was used as one of the main arguments.

2. No information gain without disturbance principle. There is a large need to get a better insight concerning the dependences between information obtained from a quantum state and its disturbance. This is of crucial importance for a deeper study of the security of quantum cryptography systems and protocols.
3. Irreversibility of quantum measurements principle. In some special cases we can, in reality, observe that quantum measurements are actually reversible. For example, in the case of teleportation, when they even "do useful work", to transform quantum information from one place to another, almost. The cases when quantum measurements are reversible need to be more explored.

4.9 Quantum Automata Theory

Two problems seem to be of the main importance here:

1. To develop quantum variants of the main classical models of quantum automata: finite automata, cellular automata, Turing machines and so on. On one side, one has to study quantum models per se and on the other side also their relations to their classical counterparts – for example, with respect to their recognition and descriptional power (see Gruska, 1999a).
2. To develop inherently quantum models of automata.

4.10 Quantum Algorithms and Communication Protocols

After the apt killing factorization and discrete logarithm algorithms of Shor and an influential algorithm of Grover, the progress in the area of the designing of quantum algorithms and networks has been steady, but not really spectacular, as was hoped for. At the same time, for the whole development of quantum information processing technology, it is of upmost importance to see clearly benefits quantum computers could provide. Much more insight is therefore needed into the real potentials of quantum computing and communication.

There have been several attempts to develop methodologies to design quantum algorithms, by amplitude amplification, see [21] for a review.

Recently, the so-called query complexity, at which one asks for the minimal number of queries to an oracle, has emerged as an important part of quantum complexity theory [14,21]. In addition, interesting relations between computational complexity, query complexity and communication complexity has been discovered – see [14] for an overview.

The emphasis seem also be shifting from the investigation of the computational problem to the investigation of the communicational problems, where one can make use of the entanglement in a much more straightforward way.

It has been shown in the area of quantum communication complexity that there are communication tasks for which quantum communication complexity provides exponentially better result than classical randomized communication complexity [44].

Let us now present some particular challenges.

1. Is there a polynomial time algorithm for the hidden subgroup problem for non-commutative groups?[13]
2. A search for zero-error quantum algorithms. Many polynomial time quantum algorithms provide their results with (small) errors. Can we find for the same problems also zero-error and still polynomial quantum algorithms? There is already a variety results in this area.
3. New basic quantum transforms need to be developed and investigated as tools to design quantum algorithms.
4. Quantum distributed computing need to be explored. In this setting it seems to be a good chance to make a significant use of entanglement.
   Design of spatially distributed quantum networks and development of quantum communication and cooperation protocols for distributed quantum computing is one of the big challenges for quantum algorithms/protocols computation/communication theory and practice.
   The basic idea is at first to distribute entanglement among distance nodes of the network and then to make use of the entanglement to make communication and computation more efficient than the classical one.
   Some of the very basic problems behind this approach have already been solved, at least theoretically, and various ways are being explored to have

[13] **Hidden subgroup problem**

Given: An (efficiently computable) function $f\colon G \rightarrow R$, where G is a finite group and R a finite set.

Promise: There exists a subgroup $G_0 \leq G$ such that f is constant and distinct on the cosets of G_0.

Task: Find a generating set for G_0 (in polynomial (in $\lg |G|$) number of calls to the oracle for f and in the overall polynomial time).

This problem is a natural generalization of several problems that have been investigated in quantum computing. For example, the Simon problem, the integer factorization problem and the discrete logarithm problem.

(a) **Order-finding problem,** $G = \mathbf{Z}$, $a \in \mathbf{N}$, $f(x) = a^x$, $x - y \in G_0 \Leftrightarrow f(x) = f(y)$, $G_0 = rk \,|\, k \in \mathbf{Z}$ for the smallest r such that $a^r = 1$. Find r.

(b) **Discrete logarithm problem,** $G = \mathbf{Z}_r \times \mathbf{Z}_r$, $a^r = 1$, $b = a^m$, $a, b \in \mathbf{N}$, $f(x, y) = a^x b^y$, $f(x_1, y_1) = f(x_2, y_2) \Leftrightarrow (x_1, y_1) - (x_2, y_2) \in G_0$. $G_0 = \{(k, -km) \,|\, k \in \mathbf{Z}_r\}$. Find G_0 (or m).

It has been shown that there is a polynomial time algorithm for the Hidden subgroup problem for the case of Abelian groups. The non-commutative case is of interest not only as a natural generalization, but also because the graph isomorphism problem falls into this category. The existence of polynomial time algorithms has been shown already for some non-commutative groups (see [21] for an overview and references), but the general case remains to be open.

efficient implementations. Quantum teleportation techniques, in combination with quantum purification and distillation techniques, allow, in principle, distribution of the entanglement through noisy channels. The range of quantum communication can be extended to very distant nodes using "quantum repeaters" that make use of quantum entanglement and quantum error correcting techniques to restore quantum signals without reading quantum information and in this way to bypass limitations imposed by quantum no-cloning theorem.

4.11 Foundational Issues

1. Non-locality is, naturally, one of the key topics in the area of very fundamental issues. There are many fundamental questions related to this problem. For example, to which extend and in which sense is quantum information global? It is therefore natural that this subject keep attracting attention of those not very happy with the current state of the understanding and of those with a hope of a fundamental contributions. Einstein's criticism of current quantum theory is still not without admirers and followers and even if the hidden variable approach did not succeed a search for other variants have not stopped. See, for example, the recent paper by Deutsch and Hayden [16]. Concerning the other recent results, let us mention the discovery of non-locality without entanglement [4].
2. A search for "new physics" for a different theories of physical world – especially those approaches motivated by QIP problems, paradigms and insights.
3. New QIP paradigms. This is a natural and long standing goal. Recently developed idea of quantum computation with continuous variables (or in infinite dimensional Hilbert spaces) falls into this framework.

4.12 Quantum Paradoxical Phenomena

There is a variety of fascinating paradoxical quantum phenomena that need to be studied in depth. Quantum non-locality is one of them. There are few others, worth to investigate.

1. Quantum teleportation. Though mathematically simple, quantum teleportation is a puzzling phenomenon. How is it possible that we can "instantaneously", within two bits, transfer an unknown quantum state? (An interesting explanation, in terms of the many world interpretation, is given by Vaidman [46]: He takes the position that teleportation procedure does not move the unknown quantum state. The state was at the Bob's place from the very beginning. The local measurement performed by Alice only splits the world in such a way that in each of the new worlds the state of Bob's particle differs from the unknown state only by one of the well known transformations, and the number of these transformations happens to be small, just 4.)

2. Counterfactual computations. They are processes by which one can learn the results of the computation without actually performing the computation – provided the possibility to perform that computation is available, even though computation itself has not been performed. (Potentials and limitations of the counterfactual computations have been recently studied by Mitchison and Jozsa [39].)

4.13 Exploring Relations and Impacts to Other Sciences

Even if the design of quantum computation and communication systems is a very important goal of the research in the area of QIP, it is not a single one. Of importance is also to identify areas outside of QIP where results of QIP theory could help to solve major problems and to make significant contributions to other sciences, and then to concentrate also on the development of the corresponding areas of QIP. Let us mention some of such problems.

1. Simulating quantum physics. Even a 10–20 qubit processor, what seems to be feasible, should be a powerful tool to simulate quantum phenomena what could be used by scientists in various areas to perform experiments. For example, in nuclear physics, material sciences, molecular biology, ...
2. A search for interfaces between physics and (quantum) information (processing). An understanding seems to be emerging that some problems of physics and physical phenomena can be elucidated by taking QIP points of view, paradigms, concepts and methods. Especially phenomena exhibiting multi-particle entanglement. Moreover, in terms of quantum information some important principles have been already formulated. For example, the "holographic principle" saying that quantum information encoded in a spatial volume can be read completely on the surface that bounds the volume. By Preskill [43], "The holographic viewpoint is particularly powerful in the case of the quantum behaviour of a black hole. The information that disappears behind the event horizon can be completely encoded on the horizon, and so can be transferred to the outgoing Hawking radiation that is emitted as the black hole evaporates. This way, the evaporation process need not destroy any quantum information."
3. Impacts on physics (theoretical and experimental).
   (a) Deepening of our understanding of quantum mechanics (especially through enhancement of our knowledge on such issues as non-locality, measurements, entanglement, decoherence and so on).
   (b) Improvement of information gathering capabilities of physical experiments [43,13]. They expect that progress in QIP can bring new strategies to perform high precision measurements (as those needed for detection of gravitational waves and other weak forces). Progress in QIP could help, for example, to develop a theory of distinguishability of superoperators (a measure of the amount of information needed to extract in order to be able to distinguished two superoperators, what could provide limitations on precision measurements), and for that they envision

and analyse utilization of even such QIP tools and methods as quantum Fourier transform, Grover's search method and superdense coding.

In general, one can expect that the ideas emerging from the QIP theory can have impact on experimental physics. Especially the understanding and utilization of the entanglement can influence experimental methods.

(c) An understanding of the behaviour of strongly-coupled many-body systems – one of the most challenging problems in quantum dynamics [43]. This can be helped by a deeper study of multi-particle entanglement.

4. Determination of the ultimate limitations for computation. This is a challenging task – both for computing and physics. As analysed by Lloyd [34], the speed with which a physical device can process information is limited by its energy (needed to switch from one state to another, distinguishable, state), and therefore on the basis of quantum mechanics one can determine that for a computer of 1 kg of mass and of 1 l of volume, the maximal ultimate speed is 2.7×10^{50} bit operations per second. It seems harder to determine the number of bits such a "ultimate laptop" can store in general (entropy limits the amount of information a computer can store), but it can be determined for a computer that has been compressed to form a black hole (to be 3.8×10^{16} bits). Estimations are based on such constants of physics as the speed of light, the Planck constant and the gravitation constant – and do not make any computer architecture assumption. (The question how black holes deal with information is an interesting problem of current physics – whether they destroy information or not – see also the holographic principle. It seems that new physics is needed to understand these problems.) The need to deal with errors can be an important factor why the above limits, based on only such characteristics of a computer as mass and volume, can be hard to achieve.

References

1. D. Bacon, D. A. Lidar, and K. B. Whaley. Robustness of decoherence-free subspaces for quantum computation. Technical report, quant-ph/9902041, 1999. 16
2. Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. Di Vincenzo, Norman Margolus, Peter W. Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates of quantum computation. *Physical Review A*, 52(5):3457–3467, 1995. 11
3. Charles H. Bennett and Peter W. Shor. Quantum information theory. *IEEE Transactions on Information Theory*, 44:2724–2742, 1998. 21
4. Charles H. Bennett, David P. DiVincenzo, Christopher A. Fuchs, Tal Moric, Eric M. Rains, Peter W. Shor, and John A. Smolin. Quantifying non-locality without entanglement. Technical report, quant-ph/9804053, 1998. 24
5. Dan Boneh and Richard J. Lipton. Quantum cryptoanalysis of hidden linear functions. In *Advances in Cryptology, Proceedings of CRYPTO '95*, pages 424–437, 1995. 9
6. P. Oscar Boykin, Tal Mor, Matthew Pulver, Vwani Roychowdhury, and Farrokh Vatan. On universal and fault-tolerant quantum computing. Technical report, quant-ph/9900054, 1999. 11, 18

7. Gilles Brassard, Richard Cleve, and Alain Tapp. The cost of exactly simulating quantum entanglement with classical communication. Technical report, quant-ph/9901035, 1999. 15
8. Shamuel L. Braunstein and H. Jeff Kimble. Exprimental quantum teleportation. *Nature*, 394:840–841, 1998. 11
9. Dagmar Bruß. Entanglement splitting of pure bipartite quantum states. Technical report, quant-ph/9902023, 1999. 15
10. Vladimír Bužek and Mark Hillery. Quantum copying: beyond the no-clonning theorem. *Physical Review A*, 54:1844–1852, 1996. 22
11. Vladimír Bužek, Samuel L. Braunstein, Mark Hillery, and Dagmar Bruß. Quantum copying: A network. *Physical Review A*, 56(5):3446–3452, 1997.
12. Guiseppe Castagnoli, Dalida Monti, and Alexander Sergienko. Symmetry, reversibility, and efficiency of quantum computation. Technical report, quant-ph/9908015, 1999. 19
13. Andrew M. Childs, John Preskill, and Joseph Renes. Quantum information and precission measurement. Technical report, quant-ph/9904021, 1999. 25
14. Richard Cleve. An introduction to quantum complexity theory. Technical report, quant-ph/9906111, 1999. 22, 22
15. Valerie Coffman, Joydep Kundu, and William K. Wooters. Distributed entanglement. Technical report, quant-ph/9907047, 1999. 15
16. David Deutsch and Patrick Hayden. Information flow in entangled quantum states. Technical report, quant-ph/9906007, 1999. 11, 24
17. David P. DiVincenzo and D. Loss. Quantum computers and quantum coherence. Technical report, quant-ph/9901137, 1999. 11
18. A. Furusawa, J. Sorenson, Shamuel L. Braunstein, Christopher A. Fuchs, H. Jeff Kimble, and E. S. Polzik. *Science*, 282:706–707, 1998. 11
19. Lior Goldenberg, Lev Vaidman, and Stephen Wiesner. Quantum gambling. *Physical Review Letters*, 82:3356–3359, 1999. 13
20. Daniel Gottesman and Isaac L. Chuang. Quantum teleportation is a universal computational primitive. Technical report, quant-ph/9908010, 1999. 11, 15, 18
21. Jozef Gruska. *Quantum computing*. McGraw-Hill, 1999.
URL: http://www.mcgraw-hill.co.uk/gruska. 3, 7, 12, 13, 13, 14, 17, 22, 22, 23
22. Jozef Gruska. Quantum challenges in descriptional complexity. In *Proceedings of the International Workshop on Descriptional Complexity*, pages 23–38, 1999.
23. Lucien Hardy. Disentangling nonlocality and teleportation. Technical report, quant-ph/9906123, 1999. 11
24. O. Hay and Asher Peres. Quantum and classical description of a measuring apparatus. Technical report, quant-ph/9712044, 1997. 20
25. Michal Horodecki, Pawel Horodecki, and Ryszard Horodecki. Separability of mixed states: necessary and sufficient conditions. *Physics Letters A*, 223:1–8, 1998. quant-ph/9605038.
26. Michal Horodecki, Pawel Horodecki, and Ryszard Horodecki. Limits for entanglement measures. Technical report, quant-ph/9908065, 1999. 14
27. Pawel Horodecki, Ryszard Horodecki, and Michal Horodecki. Entanglement and thermodynamical analogies. *Acta Physica Slovaca*, 48(3):141–156, 1998. 16
28. Michal Horodecki, Pawel Horodecki, and Ryszard Horodecki. Mixed-state entanglement and distillation: is there a “bound” entanglement in nature? Technical report, quant-ph/9801069, 1998. 14, 15
29. B. E. Kane. A Si-based nuclear spin quantum computer. *Nature*, 393:133–13?, 1998. 11

30. Peter L. Knight, A. Beige, S. Bose, S. F. Huelga, M. B. Plenio, and V. Vedral. *Physics Review A*, 59:2468–, 1999. 16
31. Emanuel Knill, Raymond Laflamme, R. Martinez, and C.-H. Tseng. A cat-state benchmark on a seven bit quantum computer. Technical report, quant-ph/9908051, 1999. 10
32. Emanuel Knill, Raymond Laflamme, and Lorenza Viola. Theory of quantum error correction for general noise. Technical report, quant-ph/9908066, 1999. 4, 17
33. D. A. Lidar, D. Bacon, and K. B. Whaley. Concatenating decoherence free subspaces with quantum error correcting codes. Technical report, quant-ph/9809081, 1998.
34. Seth Lloyd. Ultimate physical limits to computation. Technical report, quant-ph/9908043, 1999. 26
35. Hoi-Kwong Lo. A simple proof of the unconditional security of quantum key distribution. Technical report, quant-ph/9904091, 1999. 13
36. Hoi-Kwong Lo and H. F. Chau. Unconditional security of quantum key distribution over arbitrarily long distance. *Science*, 283:2050–2056, 1999.
37. Hoi-Kwong Lo and Sandu Popescu. The classical communication costs of entanglement manipulation. is entaglement an inter-convertible resource? Technical report, quant-ph/9902045, 1999. 15
38. Dominic C. Mayers and Andrew C.-C. Yao Unconditional security in quantum cryptography. Technical report, quant-ph/9802025, 1998. 13
39. Graeme Mitchison and Richard Jozsa. Counterfactual computation. Technical report, quant-ph/9906026, 1999. 25
40. Michael Nielsen Conditions for a class of entangled transformatios. *Physical Review Letters*, 83(2):436–439, 1998. 15
41. Arun Kumar Pati. Minimum cbits required to transmit a qubit. Technical report, quant-ph/9907022, 1999. 11
42. Asher Peres. Separability criterion for density matrices. *Physical Review Letters*, 77:1413–1415, 1996a. 14
43. John Preskill. Quantum information and physics: some future directions. Technical report, quant-ph/9904022, 1999. 25, 25, 26
44. R. Raz. Exponential separation of quantum and classical communication complexity. In *Proceedings of the 31th ACM STOC*, pages 358–367j, 1999. 23
45. John Reif. Alternative computational models: a comparison of biomolecular and quantum computing. In *Proceedings of 18th International Conference on Foundations of Software technology and Theoretical Computer Science*, pages 102–121, 1998. For an extended version see http://www.cs.duke/ reif/paper/altcomp.ps. 19, 20, 20
46. Lev Vaidman. Teleportation: dream or reality? Technical report, quant-ph/9810089, 1998. 11, 11, 24
47. Paul Vitanyi. Two approaches to the quantitative definition of information in an individual pure quantum state. Technical report, quant-ph/9907035, 1999. 20

Stability of Approximation Algorithms for Hard Optimization Problems

Juraj Hromkovič

Dept. of Computer Science I (Algorithms and Complexity), RWTH Aachen
Ahornstraße 55, 52056 Aachen, Germany
jh@I1.informatik.rwth-aachen.de

Abstract. To specify the set of tractable (practically solvable) computing problems is one of the few main research tasks of theoretical computer science. Because of this the investigation of the possibility or the impossibility to efficiently compute approximations of hard optimization problems becomes one of the central and most fruitful areas of recent algorithm and complexity theory. The current point of view is that optimization problems are considered to be tractable if there exist polynomial-time randomized approximation algorithms that solve them with a reasonable approximation ratio. If a optimization problem does not admit such a polynomial-time algorithm, then the problem is considered to be not tractable.

The main goal of this paper is to relativize this specification of tractability. The main reason for this attempt is that we consider the requirement for the tractability to be strong because of the definition of the complexity as the "worst-case" complexity. This definition is also related to the approximation ratio of approximation algorithms and then an optimization problem is considered to be intractable because some subset of problem instances is hard. But in the practice we often have the situation that the hard problem instances do not occur. The general idea of this paper is to try to partition the set of all problem instances of a hard optimization problem into a (possibly infinite) spectrum of subclasses according to their polynomial-time approximability. Searching for a method enabling such a fine problem analysis (classification of problem instances) we introduce the concept of stability of approximation. To show that the application of this concept may lead to a "fine" characterization of the hardness of particular problem instances we consider the traveling salesperson problem and the knapsack problem.

1 Introduction

Historically, the first informally formulated questions about computability and decidability dates back at the turn of the century, when the great mathematican David Hilbert asked for the possibility to encode all mathematical problems in some suitable formalism and to determine (decide) their truth by an algorithm. The landmark paper by Kurt Gödel [11] showed that this is impossible. Moreover, this paper led to the formal definition of the notion of algorithm and so put the

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 29–47, 1999.

fundamentals of theoretical computer science. So, it was possible to formaly pose questions of the following kind:

"Is a given problem algorithmically solvable (computable, decidable)?"

This led to the development of the computability theory (for a nice overview see [22]), where mathematical methods determining the existence or non-existence of algorithms for different mathematical problems were created. With the development of computer technologies a new, more involved question than the original one becomes the central one:

"If a problem is computable, how much physical (computer) work is necessary and sufficient to algorithmically solve the problem?"

This question initialized the origin of the algorithm and complexity theory in the sixties. Measuring the complexity of algorithms the question "What is computable?" was replaced by the question

"What is practically computable?"

But to answer this question one has first to specify (formalize) the meaning of the notion *"practically computable"*, called **tractable** in the current literature. The history of the development of complexity and algorithm theory can be more or less viewed as the history of the development of our opinion on the specification of tractability.

The first specification was done already in the sixties, when people decided to consider a computing problem to be tractable if there exists an algorithm that solves the problem in polynomial-time in the input length. A problem is called intractable if it does not admit any polynomial-time algorithm. The main reasons for this decision were the following two:

(i) (*a more theoretical reason*) The class of polynomial-time computations is robust in that sense that its specification is independent on the computing model (formalism) chosen, if the model is reasonable for the complexity measurement.[1]

(ii) (*a more practical reason*) If a problem had a polynomial-time algorithm, then usually the people were able to find an at most $O(n^3)$ time algorithm for it. Such an algorithm is really practical because one can use it to solve problem instances of sizes that realistically appear in the practice. On the other hand, if the complexity of an algorithm is 2^n, where n is the input size, then already for realistic input size $n = 100$ 2^n is a 31-digit number. Since the number of microseconds since the "Big Bang" should has 24 digits, it is clear that there is no chance to apply this algorithm.

Considering this classification (specification of tractability) the main problem becomes to find a method for proving that a computing problem does not admit

[1] Note, that this kind of robustness is a reasonable requirement that should be fulfilled for every attempt to define the class of tractable problems.

any polynomial-time algorithm, i.e., that it is intractable. Because no mathematical proof of intractability for a language from NP has been realized up till now, and the most interesting problems in practice have their decision versions in NP, Cook introduced the concept of NP-completeness in his seminal paper [9]. This concept enabled, under the assumpion P $\neq$ NP, to prove intractability of thousands of computing problems.

Immediately after introducing NP-hardness (completeness) [9] as a concept for proving intractability of computing problems [16], the following question has been posed:

> *"If an optimization problem does not admit any efficiently computable optimal solution, is there a possibility to efficiently compute at least an approximation of the optimal solution?"*

Several researchers (see, for instance [15,17,7,14]) provided already in the middle of the seventies a positive answer for some optimization problems. It may seem to be a fascinating effect if one jumps from the exponential complexity (a huge inevitable amount of physical work) to the polynomial complexity (tractable amount of physical work) due to a small change in the requirement – instead of an exact optimal solution one forces a solution whose quality differs from the quality of an optimal solution at most by $\varepsilon \cdot 100$ % for some ε. This effect is very strong, especially, if one considers problems for which this approximation concept works for any small relative difference ε (see the concept of approximation schemes in [14,19,21,4]).

On the other hand we know several computing problems that admit efficient polynomial-time randomized algorithms, but no polynomial-time deterministic algorithm is known for them.[2] Usually the randomized algorithms are practical because the probability to compute the right output can be increased to $1 - \delta$ for any small $\delta > 0$.

These are the reasons why currently optimization problems are considered to be tractable if there exist randomized polynomial-time approximation algorithms that solve them with a reasonable approximation ratio. In what follows an α-approximation algorithm for a minimization [maximization] problem is an algorithm that provides feasible solutions whose cost divided by the cost of optimal solutions is at most α [is at least $\frac{1}{\alpha}$].

There is also another possibility to jump from NP to P. Namely, to consider the subset of inputs with a special, nice property instead of the whole set of inputs for which the problem is well-defined. A nice example is the Traveling Salesperson Problem (TSP). TSP is not only NP-hard, but also the search of an approximation solution for TSP is NP-hard for every ε.[3] But if one considers TSP for inputs satisfying the triangle inequality (so called Δ-TSP), one can even design an $\frac{3}{2}$-approximation algorithm [7]. The situation is still more

[2] Currently, we do not know any randomized bounded-error polynomial-time algorithm for an NP-hard problem.

[3] In fact, under the assumtion P $\neq$ NP, there is no $p(n)$-approximation algorithm for TSP for any polynomial p.

interesting, if one considers the Euclidean TSP, where the distances between the nodes correspond to the distances in the Euclidean metrics. The Euclidean TSP is NP-hard [20], but for every small $\varepsilon > 0$ one can design a polynomial-time $(1+\varepsilon)$-approximation algorithm [2,3,18].[4]

Exactly this second approach for jumping from NP to P is the reason why we propose again to revise the notion of tractability of optimization problems. This approach shows that the main drawback of the current definition of tractability is in the definition of complexity and approximality in the worst-case manner. Because of some hard problem instances a problem is considered to be intractable. But if one can specify the border between the hard problem instances and the easy ones, and one can efficiently decide the membership of a given instance to one of this two classes, then this computing problem may finally be considered to be tractable in several applications. Our general idea is to try to split the set of all input sequences of the given problem into possible infinitely many subclasses according to the hardness of their polynomial-time approximability. To provide a method that is able to achieve this goal, we introduce the concept of approximation stability.

Informally, one can describe the idea of our concept by the following scenario. One has an optimization problem P for two sets of input instances L_1 and L_2, $L_1 \subset L_2$. For L_1 there exists a polynomial-time α-approximation algorithm A, but for L_2 there is no polynomial-time γ-approximation algorithm for any $\gamma > 0$ (if NP is not equal to P). We pose the following question: Is the algorithm A really useful for inputs from L_1 only? Let us consider a distance measure M in L_2 determining the distance $M(x)$ between L_1 and any $x \in L_2$. Now, one can consider an input $x \in L_2 - L_1$, with $M(x) \leq k$ for some positive real k. One can look for how "good" the algorithm A is for the input $x \in L_2 - L_1$. If for every $k > 0$ and every x with the distance at most k to L_1, A computes an $\delta_{\alpha,k}$ approximation of an optimal solution for x ($\delta_{\alpha,k}$ is considered to be a constant depending on k and α only), then one can say that A is "(approximation) stable" according to M.

Obviously, the idea of the concept of approximation stability is similar to that of stability of numerical algorithms. But instead of observing the size of the change of the output value according to a small change of the input value, we look for the size of the change of the approximation ratio according to a small change in the specification (some parameters, characteristics) of the set of input instances considered. If the exchange of the approximation ratio is small for every small change in the specification of the set of input instances, then we have a stable algorithm. If a small change in the specification of the set of input instances causes an essential (including the size of the input instances) increase of the relative error, then the algorithm is unstable.

[4] Obviously, we know a lot if similar examples where with restricting the set of inputs one crosses the border between decidability and undecidability (Post Correspondence Problem) or the border between P and NP (SAT and 2-SAT, or vertex cover problem).

The concept of stability enables to show positive results extending the applicability of known approximation algorithms. As we shall see later, the concept motivates us to modify an unstable algorithm A in order to get a stable algorithm B that achieves the same approximation ratio on the original set of input instances as A has, but B can be successfully used also outside of the original set of input instances. This concept seems to be useful because there are many problems for which an additional assumption on the "parameters" of the input instances leads to an essential decrease of the hardness of the problem. Thus, such "effects" are the starting points for trying to partition the whole set of input instances into a spectrum of classes according to approximability.

This paper is organized as follows: In Section 2 we present our concept of approximation stability. Sections 3 and 4 illustrates the usefulness of this concept by applying it to TSP and the knapsack problem respectively. Section 5 is devoted to a general discussion about possible applications of this concept.

2 The Concept of the Stability of Approximation

We assume that the reader is familiar with the basic concepts and notions of algorithmics and complexity theory as presented in standard textbooks like [4,10,13,21,23]. Next, we give a new (a little bit revised) definition of the notion of an optimization problem. The reason to do this is to obtain the possibility to study the influence of the input sets on the hardness of the problem considered. Let $\mathbb{N} = \{0, 1, 2, \dots\}$ be the set of nonnegative integers, and let $\mathbb{R}^+$ be the set of positive reals.

Definition 1. *An **optimization problem** U is an 7-tuple $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$, where*

*(i) Σ_I is an alphabet called **input alphabet**,*
*(ii) Σ_O is an alphabet called **output alphabet**,*
*(iii) $L \subseteq \Sigma_I^*$ is a language over Σ_I called the **language of consistent inputs**,*
*(iv) $L_I \subseteq L$ is a language over Σ_I called the **language of actual inputs**,*
*(v) $\mathcal{M}$ is a function from L to $2^{\Sigma_O^*}$, where, for every $x \in L$, $\mathcal{M}(x)$ is called the **set of feasible solutions** for the input x,*
*(vi) cost is a function, called **cost function**, from $\bigcup_{x \in L} \mathcal{M}(x) \times L_I$ to $\mathbb{R}^+$,*
(vii) goal $\in$ {minimum, maximum}.

For every $x \in L$, we define

$$\boldsymbol{Output_U(x)} = \{y \in \mathcal{M}(x) \mid cost(y) = goal\{cost(z) \mid z \in \mathcal{M}(x)\}\},$$

and

$$\boldsymbol{Opt(x)} = cost(y) \text{ for some } y \in Output_U(x).$$

Clearly, the meaning for Σ_I, Σ_O, $\mathcal{M}$, *cost* and *goal* is the usual one. L may be considered as a set of consistent inputs, i.e., the inputs for which the optimization problem is consistently defined. L_I is the set of inputs considered and only these inputs are taken into account when one determines the complexity of the optimization problem U. This kind of definition is useful for considering the complexity of optimization problems parametrized according to their languages of actual inputs. In what follows ***Language(U)*** denotes the language L_I of actual inputs of U.

To illustrate Definition 1 consider the Knapsack Problem (KP). In what follows ***bin(u)*** denotes the integer binary coded by the string $u \in \{0,1\}^*$. The input of KP consists of $2n+1$ integers $w_1, w_2, \ldots, w_n, b, c_1, c_2, \ldots, c_n, n \in \mathbb{N}$. So, one can consider $\Sigma_I = \{0,1,\#\}$ with the binary coding of integers and # for ",". The output is a vector $x \in \{0,1\}^n$, and so we set $\Sigma_O = \{0,1\}$. $L = \{0,1\}^* \cdot \bigcup_{i=0}^{\infty} (\#\{0,1\}^*)^{2i}$. We speak about the Simple Knapsack Problem (SKP) if $w_i = c_i$ for every $i = 1, \ldots, n$. So, we can consider $L_I = \{z_1\#z_2\#\ldots\#z_n\#b\#z_1\#z_2\#\ldots\#z_n \mid b,\ z_i \in \{0,1\}^* \text{ for } i = 1,\ldots,n,\ n \in \mathbb{N}\}$ as a subset of L. $\mathcal{M}$ assigns to each $I = y_1\#\ldots\#y_n\#b\#u_1\#\ldots\#u_n$ the set of words $\mathcal{M}(I) = \{x = x_1x_2\ldots x_n \in \{0,1\}^n \mid \sum_{i=1}^{n} x_i \cdot bin(y_i) \leq bin(b)\}$. For every $x = x_1 \ldots x_n \in \mathcal{M}(I)$, $cost(x) = \sum_{i=1}^{n} x_i \cdot bin(u_i)$. The goal is maximum. So, $KP = (\Sigma_I, \Sigma_O, L, L, \mathcal{M}, cost, goal)$, and $SKP = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$.

Definition 2. *Let $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ be an optimization problem. We say that an algorithm A is a **consistent algorithm for U** if, for every input $x \in L_I$, A computes an output $A(x) \in \mathcal{M}(x)$. We say that **A solves U** if, for every $x \in L_I$, A computes an output $A(x)$ from $Output_U(x)$. The time complexity of A ist defined as the function*

$$Time_A(n) = \max\{Time_A(x) \mid x \in L_I \cap \Sigma_I^n\}$$

from $\mathbb{N}$ to $\mathbb{N}$, where $Time_A(x)$ is the length of the computation of A on x.

Next, we give the definitions of standard notions in the area of approximation algorithms (see e.g. [8,13]).

Definition 3. *Let $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ be an optimization problem, and let A be a consistent algorithm for U. For every $x \in L_I$, the **relative error $\varepsilon_A(x)$** is defined as*

$$\boldsymbol{\varepsilon_A(x)} = \frac{|cost(A(x)) - Opt(x)|}{Opt(x)}.$$

*For any $n \in \mathbb{N}$, we define **the relative error of A***

$$\boldsymbol{\varepsilon_A(n)} = \max\{\varepsilon_A(x) \mid x \in L_I \cap \Sigma_I^n\}.$$

*For every $x \in L_I$, the **approximation ratio $R_A(x)$** is defined as*

$$\boldsymbol{R_A(x)} = \max\left\{\frac{cost(A(x))}{Opt(x)}, \frac{Opt(x)}{cost(A(x))}\right\} = 1 + \varepsilon_A(x).$$

For any $n \in \mathbb{N}$, *we define the* ***approximation ratio of A*** *as*

$$\boldsymbol{R_A(n)} = \max\{R_A(x) \mid x \in L_I \cap \Sigma_I^n\}.$$

For any positive real $\delta > 1$, *we say that* A *is an* $\boldsymbol{\delta}$***-approximation algorithm for U*** *if* $R_A(x) \leq \delta$ *for every* $x \in L_I$.

For every function $f : \mathbb{N} \to \mathbb{R}^+$, *we say that* A *is a* $\boldsymbol{f(n)}$***-approximation algorithm for U*** *if* $R_A(n) \leq f(n)$ *for every* $n \in \mathbb{N}$.

The best what can happen for a hard optimization problem U is that one has a polynomial-time $(1+\varepsilon)$-approximation algorithm for U for any $\varepsilon > 0$. In that case we call this collection of approximation algorithms a ***polynomial-time approximation scheme*** (**PTAS**) ***for U***. A nicer definition of a PTAS than the above one is the following one. An algorithm B is a PTAS for an optimization problem U if B computes an output $B(x,\varepsilon) \in \mathcal{M}(x)$ for every input $(x,\varepsilon) \in Language_U \times \mathbb{R}^+$ with

$$\frac{|cost(B(x,\varepsilon)) - Opt(x)|}{Opt(x)} \leq \varepsilon$$

in time polynomial according to $|x|$. If the time complexity of B can be bounded by a function that is polynomial in both $|x|$ and ε^{-1}, then we say that B is a ***fully polynomial-time approximation schema*** **(FPTAS)** for U.

Now, we define the complexity classes of optimization problems in the usual way (see e.g. [13,19]).

Definition 4.

$$\begin{aligned}\boldsymbol{NPO} = \{\ & U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal) \mid \\ & U \textit{ is an optimization problem}, \quad L, L_I \in P; \\ & \textit{for every } x \in L, \mathcal{M}(x) \in P; cost \textit{ is computable in polynomial time}\},\end{aligned}$$

For every optimization problem $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$, *the* ***underlying language of U*** *is*

$$Under_U = \{(w,k) \mid w \in L_I, k \in \mathbb{N} - \{0\}, Opt_U(w) \leq k\}$$

if goal = *maximum. Analogously, if goal* = *minimum*

$$Under_U = \{(w,r) \mid w \in L_I,\ r \in \mathbb{N} - \{0\},\ Opt_U(w) \geq r\}.$$

$\boldsymbol{PO} = \{U \in NPO \mid Under_U \in P\}$
$\boldsymbol{APX} = \{U \in NPO \mid$ *there exists an* ε*-approximation algorithm for* U *for some* $\varepsilon \in \mathbb{R}^+\}$.

In order to define the notion of stability of approximation algorithms we need to consider something like a distance between a language L and a word outside L.

Definition 5. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *and* $\overline{U} = (\Sigma_I, \Sigma_O, L, L, \mathcal{M}, cost, goal)$ *be two optimization problems with* $L_I \subset L$. *A* ***distance function for U according to*** $\boldsymbol{L_I}$ *is any function* $h : L \to \mathrm{I\!R}^+$ *satisfying the properties*

(i)

$$h(x) = 0 \textit{ for every } x \in L_I.$$

(ii)

$$h \textit{ can be computed in polynomial time.}$$

We define, for any $r \in \mathrm{I\!R}^+$,

$$\boldsymbol{Ball_{r,h}(L_I)} = \{w \in L \mid h(w) \leq r\}.$$

Let A *be a consistent algorithm for* $\overline{U}$, *and let* A *be an* ε*-approximation algorithm for* U *for some* $\varepsilon \in \mathrm{I\!R}^+$. *Let* p *be a positive real. We say that* A *is* ***p-stable according to h*** *if, for every real* $0 \leq r \leq p$, *there exists a* $\delta_{r,\varepsilon} \in \mathrm{I\!R}^+$ *such that* A *is an* $\delta_{r,\varepsilon}$*-approximation algorithm for* $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$.[5]

A is ***stable according to h*** *if A is p-stable according to h for every* $p \in \mathrm{I\!R}^+$. *We say that A is* ***unstable according to h*** *if A is not p-stable for any* $p \in \mathrm{I\!R}^+$.

For every positive integer r, *and every function* $f_r : \mathrm{I\!N} \to \mathrm{I\!R}^+$ *we say that* A *is* $\boldsymbol{(r, f(n))}$***-quasistable according to h*** *if* A *is an* $f_r(n)$*-approximation algorithm for* $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$.

We see that the existence of a stable c-approximation algorithm for U immediately implies the existence of a $\delta_{r,c}$-approximation algorithm for U_r for any $r > 0$. Note, that applying the stability to PTASs one can get two different outcomes. Consider a PTAS A as a collection of polynomial-time $(1+\varepsilon)$-approximation algorithms A_ε for every $\varepsilon \in \mathrm{I\!R}^+$. If A_ε is stable according to a distance measure h for every $\varepsilon > 0$, then we can obtain either

(i) a PTAS for $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$ for every $r \in \mathrm{I\!R}^+$ (this happens, for instance, if $\delta_{r,\varepsilon} = 1 + \varepsilon \cdot f(r)$, where f is an arbitrary function), or

(ii) a $\delta_{r,\varepsilon}$-approximation algorithm for U_r for every $r \in \mathrm{I\!R}^+$, but no PTAS for U_r for any $r \in \mathrm{I\!R}^+$ (this happens, for instance, if $\delta_{r,\varepsilon} = 1 + r + \varepsilon$).

To capture these two different situations we introduce the notion of "superstability" as follows:

Definition 6. *Let* $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ *and* $\overline{U} = (\Sigma_I, \Sigma_O, L, L, \mathcal{M}, cost, goal)$ *be two optimization problems with* $L_I \subset L$. *Let* h *be a distance function for* $\bar{U}$ *according to* L_I, *and let* $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$ *for every* $r \in \mathrm{I\!R}^+$. *Let* $A = \{A_\varepsilon\}_{\varepsilon>0}$ *be a PTAS for* U.

If, for every $r > 0$ *and every* $\varepsilon > 0$, A_ε *is a* $\delta_{r,\varepsilon}$*-approximation algorithm for* U_r, *we say that the PTAS* A *is* **stable according to** $\boldsymbol{h}$.

[5] Note, that $\delta_{r,\varepsilon}$ is a constant depending on r and ε only.

If $\delta_{r,\varepsilon} \leq f(\varepsilon) \cdot g(r)$, *where*

(i) f *and* g *are some functions from* $\mathrm{I\!R}^{\geq 0}$ *to* $\mathrm{I\!R}^{+}$, *and*
(ii) $\lim_{\varepsilon \to 0} f(\varepsilon) = 0$,

then we say that the PTAS A is **super-stable according to *h*.**

Remark 1. If A is a super-stable (according to a distance function h) PTAS for an optimization problem $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$, then A is a PTAS for the optimization problem $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$ for any $r \in \mathrm{I\!R}^+$.

One may see that the notions of stability can be useful for answering the question how broadly a given approximation algorithm is applicable. So, if one is interested in positive results then one is looking for a suitable distance measure that enables to use the algorithm outside the originally considered set of inputs. In this way one can search for the border of the applicability of the given algorithm. If one is interested in negative results then one can try to show that for any reasonable distance measure the considered algorithm cannot be extended to work for a much larger set of inputs than the original one. In this way one can search for fine boundaries between polynomial approximability and polynomial non-approximability.

3 Stability of Approximation and Traveling Salesperson Problem

We consider the well-known TSP problem that is in its general form very hard for approximation. But if one considers complete graphs in which the triangle inequality holds, then we have the two following approximation algorithms for the Δ-TSP.

Algorithm ST (Spanning Tree)

Input: a complete graph $G = (V, E)$, and a cost function $c : E \to \mathrm{I\!N}^+$ satisfying the triangle inequality ($c(\{u, v\}) \leq c(\{v, w\}) + c(\{w, u\})$ for all three pairwise different $u, v, w \in V$).
Step 1: Construct a minimal spanning tree T of G according to c.
Step 2: Choose an arbitrary vertex $v \in V$. Realize Depth-first-search of T from v, and order the vertices in order in that they are visited. Let H be the resulting sequence.
Output: The Hamiltonian tour $\bar{H} = H, v$.

Christofides Algorithm

Input: a complete graph $G = (V, E)$, and a cost function $c : E \to \mathbb{N}^+$ satisfying the triangle inequality.
Step 1: Construct a minimal spanning tree T of G according to c.
Step 2: $S := \{v \in V \mid deg_T(v) \text{ is odd}\}$.
Step 3: Compute a minimum-weight perfect matching M on S in G.
Step 4: Create the multigraph $G' = (V, E(T) \cup M)$ and construct an Euler tour w in G'.
Step 5: Construct a Hamiltonian tour H of G by shortening w {realize it by removing all repetitions of the occurences of every vertex in w, in one run via w from the left to the right}.
Output: H

It is well known that the algorithm ST and the Christofides algorithm are α-approximation algorithms for the Δ-TSP with $\alpha = 2$ and $\alpha = \frac{3}{2}$ respectively. It can be simply observed that both algorithms are consistent for the general TSP. Since the triangle inequality is crucial for the approximability of the problem instances of Δ-TSP, we consider the following distance functions.

We define for every $x \in L$,

$$dist(x) = \max\left\{0, \max\left\{\frac{c(\{u,v\})}{c(\{u,p\}) + c(\{p,v\})} - 1 \,\middle|\, u, v, p \in V_x\right\}\right\},$$

and

$$distance(x) = \max\left\{0, \max\left\{\frac{c(\{u,v\})}{\sum_{i=1}^{m} c(\{p_i, p_{i+1}\})} - 1 \,\middle|\, u, v \in V_x, \right.\right.$$

$$\left.\left. \text{and } u = p_1, p_2, \ldots, p_{m+1} = v \text{ is a simple path between } u \text{ and } v \text{ in } G_x \right\}\right\}$$

We observe that *dist* and *distance* measure the “degree” of violating the triangle inequality in two different ways. Let $x = (G, c)$ be an input instance of TSP. For the simplicity we consider the size of x as the number of nodes of G instead of the real length of the code of x over Σ_I. The inequality $dist(G, c) \leq r$ implies

$$c(\{u,v\}) \leq (1 + r) \cdot [c(\{u,w\}) + c(\{w,v\})]$$

for all pairwise different vertices u, v, w of G. The measure *distance* involves a much harder requirement than *dist*. If $distance(G, c) \leq r$, then $c(\{u,v\})$ may not be larger than $(1 + r)$ times the cost of any path between u and v[6]. The next results show that these two distance functions are really different because the approximation algorithms for Δ-TSP considered above are stable according to *distance* but not according to *dist*.

Lemma 1. *The algorithm SP, and the Christofides algorithm are stable according to distance.*

[6] than the cost of the shortest path between u and v

Proof. We present the proof for the algorithm SP only. Let $x \in Ball_{r,distance}(L_I)$ for an $r \in \mathbb{R}^+$. Let D_x be the Eulerian tour corresponding to the moves of DFS in Step 2. Observe that D_x goes twice via every edge of the minimal spanning tree T. Since the cost of T is smaller than the cost of any optimal Hamiltonian tour, the cost of D_x is at most twice the cost of an optimal Hamiltonian tour. Let $H_x = v_1, v_2, \ldots, v_n, v_1$ be the resulting Hamiltonian tour. Obviously, D_x can be written as $v_1 P_1 v_2 P_2 v_3 \ldots v_n P_n v_1$, where P_i is a path between v_i and $v_{(i+1) \bmod n}$ in D_x. Since $c(\{v_i, v_{(i+1) \bmod n}\})$ is at most $(1+r)$ times the cost of the path $v_i P_i v_{(i+1) \bmod n}$ for all $i \in \{1, 2, \ldots, n\}$, the cost for H_x is at most $(1+r)$ times the cost for D_x. Since the cost of D_x ist at most $2 \cdot Opt(x)$, the algorithm SP is a $(2 \cdot (1+r))$-approximation algorithm for $(\Sigma_I, \Sigma_O, L, Ball_{r,distance}(L_I), \mathcal{M}, cost, minimum)$. □

The next result shows that our approximation algorithms provide a very week approximation for input instances of Δ_{1+r}-TSP, where Δ_{1+r}-TSP $= (\Sigma_I, \Sigma_O, L, Ball_{r,dist}(L_\Delta), \mathcal{M}, cost, minimum)$.

Lemma 2. *For every $r \in \mathbb{R}^+$, the Christofides algorithm is $\left(r, \frac{3}{2}(1+r)^{\lceil \log_2 n \rceil}\right)$-quasistable for dist, and the algorithm ST is $\left(r, 2 \cdot (1+r)^{\lceil \log_2 n \rceil}\right)$-quasistable for dist.*

Proof. Again, we realize the proof for the algorithm ST only. Let $x = (G, c) \in Ball_{r,dist}(L_\Delta)$ for an $r \in \mathbb{R}^+$. Let T, D_x, and H_x have the same meaning as in the proof of Lemma 1. The crucial idea is the following one. To exchange a path v, P, u of a length m, $m \in \mathbb{N}^+$, for the edge $\{v, u\}$ one can proceed as follows. For any $p, s, t \in V(G)$ one can exchange the path p, s, t for the edge $\{p, t\}$ by the cost increase bounded by the multiplicative constant $(1+r)$. This means that reducing the length m of a path to the length $\lceil m/2 \rceil$ increases the cost of the connection between u and v by at most $(1+r)$-times. After at most $\lceil \log_2 m \rceil$ such reduction steps one reduces the path v, P, u to the path v, u and

$$cost(u, v) = c(\{v, u\}) \leq (1+r)^{\lceil \log_2 m \rceil} \cdot cost(v, P, u) \ .$$

Following the argumentation of the proof of Lemma 1 we have $c(D_x) \leq 2 \cdot Opt(x)$. Since $m \leq n$ for the length m of any path of D_x exchanged by a single edge, we obtain

$$c(H_x) \leq (1+r)^{\lceil \log_2 n \rceil} \cdot c(D_x) \leq 2 \cdot (1+r)^{\lceil \log_2 n \rceil} \cdot Opt(x) \ .$$

□

Now, we show that our approximation algorithms are really unstable according to *dist*. To show this, we construct an input, for which the Christofides algorithm provides a very poor approximation.

We construct a weighted complete graph from $Ball_{r,dist}(L_I)$ as follows. We start with the path $p_0, p_1, \ldots, p_n$ for $n = 2^k$, $k \in \mathbb{N}$, where every edge $\{p_i, p_{i+1}\}$ has the cost 1. Then we add edges (p_i, p_{i+2}) for $i = 0, 1, \ldots, n-2$ with the cost $2 \cdot (1+r)$. Generally, for every $m \in \{1, \ldots, \log_2 n\}$, we define $c(\{p_i, p_{i+2^m}\}) =$

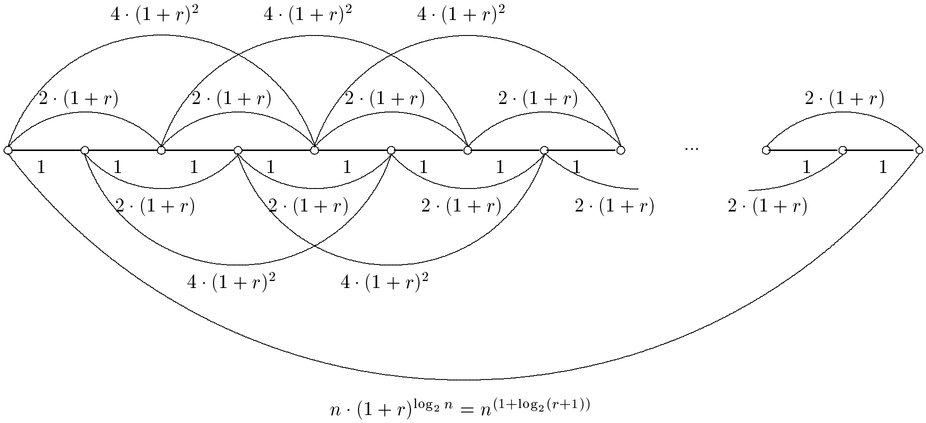


Fig. 1.

$2^m \cdot (1+r)^m$ for $i = 0, \ldots, n - 2^m$. For all other edges one can take the maximal possible cost in such a way that the constructed input is in $Ball_{r,dist}(L_I)$.

Let us have a look on the work of our algorithms on this input. There is only one minimal spanning tree that corresponds to the path containing all edges of the weight 1. Since every path contains exactly two nodes of odd degrees, the Euler tour constructed by the Christofides algorithm is the cycle $D = p_0, p_1, p_2, \ldots, p_n, p_0$ with the n edges of weight 1 and the edge of the maximal weight $n \cdot (1+r)^{\log_2 n} = n^{1+\log_2(1+r)}$. Since the Euler tour is a Hamiltonian tour, the output of the Christofides algorithm is unambigously the cycle $p_0, p_1, \ldots, p_n, p_0$ with the cost $n + n(1+r)^{\log_2 n}$. Observe, that the algorithm ST computes as output the same tour if the depth-first-search started from p_0 or from p_n. But the optimal tour is

$$T = p_0, p_2, p_4, \ldots, p_{2i}, p_{2(i+1)}, \ldots, p_n, p_{n-1}, p_{n-3}, \ldots, p_{2i+1}, p_{2i-1}, \ldots, p_3, p_1, p_0.$$

This tour contains two edges $\{p_0, p_1\}$ and $\{p_{n-1}, p_n\}$ of the weight 1 and all $n-2$ edges of the weight $2 \cdot (1+r)$. Thus, $Opt = cost(T) = 2 + 2 \cdot (1+r) \cdot (n-2)$, and

$$\begin{aligned} \frac{cost(D)}{cost(T)} &= \frac{n + n^{1+\log_2(1+r)}}{2 + 2 \cdot (1+r) \cdot (n-2)} \\ &\geq \frac{n^{1+\log_2(1+r)}}{2 \cdot n \cdot (1+r)} = \frac{n^{\log_2(1+r)}}{2 \cdot (1+r)}. \end{aligned}$$

So, we have proved the following result.

Lemma 3. *For every $r \in \mathbb{R}^+$, if the Christofides algorithm (the algorithm ST) is $(r, f(n,r))$-quasistable for dist, then $f(n,r) \geq n^{\log_2(1+r)}/2 \cdot (1+r)$.*

Corollary 1. *The Christofides algorithm and the algorithm ST are unstable for dist.*

The results above show that the Christofides algorithm can be useful for a much larger set of inputs than the original input set. The key point is to find a suitable distance measure. In our case this measure is *distance* but not *dist*. An interesting question is whether one can modify these algorithms in such a way that the modified versions would be stable for *dist*.

Surprisingly, this is possible for both algorithms. To make the Christofides algorithm stable according to *dist* one first takes a minimal perfect path matching instead of an minimal perfect matching. In this way the cost of the Euler tour w remains at most $\frac{3}{2}\cdot$ the cost of an optimal Hamiltonian tour for any input instance of TSP. Secondly, we need a special procedure to produce a Hamiltonian tour H by shortening paths of w of the length at most 4 by an edge of H. Another possibility is to modify the ST algorithm. As we have observed in the example at Figure 1, the main problem is in the fact that shortening a path of a length m can increase the cost of this path by the multiplicative factor $(1+r)^{\log_2 m}$. To modify the ST algorithm one has to change the DFS-order of the vertices of the minimal spanning tree by another order, that shortens parts of the Euclid tour with length at most 3. There are too many technicalities to be able to explain these algorithms here into complete details. So, we present the final results achieved in the recent papers [1,5,6] only.

Theorem 1. *For every $\beta > 1$,*

(i) Δ_β-TSP can be approximated in polynomial-time within the approximation ratio $\min\left\{4\beta, \frac{3}{2}\beta^2\right\}$, *and*

(ii) unless P=NP, Δ_β-TSP cannot be approximated within approximation ratio $1+\varepsilon\cdot\beta$ *for some $\varepsilon > 0$.*

4 Superstability and Knapsack Problem

In this section we do not try to present any new result. We only use the known PTASs for the Knapsack problem (KP) to illustrate the transparency of the approximation stability point of view on their development. First we show that the original PTAS for the simple Knapsack problem (SKP) is stable, but not superstable, according a reasonable distance function. So, the application of this PTAS for KP leads to an approximation algorithm, but not to a PTAS. Then, we show that the known PTAS for KP is a simple modification of the PTAS for SKP, that makes this PTAS superstable according to our distance function.

The well-known PTAS for SKP works as follows:

PTAS SKP

Input: positive integers $w_1, w_2, \ldots, w_n, b$ for some $n \in \mathbb{N}$ and some positive real number $1 > \varepsilon > 0$.

Step 1: Sort $w_1, w_2, \ldots, w_n$. For simplicity we may assume $w_1 \geq w_2 \geq \cdots \geq w_n$.
Step 2: Set $k = \lceil 1/\varepsilon \rceil$.
Step 3: For every set $T = \{i_1, i_2, \ldots, i_l\} \subseteq \{1, 2, \ldots, n\}$ with $|T| = l \leq k$ and $\sum_{i \in T} w_i \leq b$, extend T to T' by using the greedy method and values $w_{i_l+1}, w_{i_l+2}, \ldots, w_n$. (The greedy method stops if $\sum_{i \in T'} w_i \leq b$ and $w_j > b - \sum_{i \in T'} w_i$ for all $j \notin T'$, $j \geq i_l$.)
Output: The best set T' constructed in Step 3.

For every given ε, we denote the above algorithm by A_ε. It is known that A_ε is an ε-approximation algorithm for SKP. Observe that it is consistent for KP. Now, we consider the following distance function *DIST* for any input $w_1, w_2, \ldots, w_n, b, c_1, \ldots, c_n$ of KP:

$$\mathit{DIST}(w_1, \ldots, w_n, b, c_1, \ldots, c_n) = \max\left\{\max\left\{\frac{c_i - w_i}{w_i} \;\middle|\; c_i \geq w_i,\ i \in \{1, \ldots, n\}\right\},\ \max\left\{\frac{w_i - c_i}{c_i} \;\middle|\; w_i \geq c_i,\ i \in \{1, \ldots, n\}\right\}\right\}.$$

Let $KP_\delta = (\Sigma_I, \Sigma_O, L, \mathit{Ball}_{\delta, \mathit{DIST}}(L_I), \mathcal{M}, \mathit{cost}, \mathit{maximum})$ for any δ. Now, we show that PTAS SKP is stable according to *DIST* but this result does not imply the existence of a PTAS for KP_δ for any $\delta > 0$.

Lemma 4. *For every $\varepsilon > 0$, and every $\delta > 0$, the algorithm A_ε is a $(1 + \varepsilon + \delta(2 + \delta) \cdot (1 + \varepsilon))$-approximation algorithm for KP_δ.*

Proof. Let $w_1 \geq w_2 \geq \cdots \geq w_n$ for an input $I = w_1, \ldots, w_n, b, c_1, \ldots, c_n$, and let $k = \lceil 1/\varepsilon \rceil$. Let $U = \{i_1, i_2, \ldots, i_l\} \subseteq \{1, 2, \ldots, n\}$ be an optimal solution for I. If $l \leq k$, then A_ε outputs an optimal solution with $\mathit{cost}(U)$ because A_ε has considered U as a candidate for the output in Step 3.

Consider the case $l > k$. A_ε has considered the greedy extension of $T = \{i_1, i_2, \ldots, i_k\}$ in Step 2. Let $T' = \{i_1, i_2, \ldots, i_k, j_{k+1}, \ldots, j_{k+r}\}$ be the greedy extension of T. Obviously, it is sufficient to show that the difference $\mathit{cost}(U) - \mathit{cost}(T')$ is small relative to $\mathit{cost}(U)$, because the cost of the output of A_ε is at least $\mathit{cost}(T')$. We distinguish the following two possibilities according to the weights of the feasable solutions U and T':

1. Let $\sum_{i \in U} w_i - \sum_{j \in T'} w_j \leq 0$.
   Obviously, for every i, $(1 + \delta)^{-1} \leq \frac{c_i}{w_i} \leq 1 + \delta$. So,

$$\mathit{cost}(U) = \sum_{i \in U} c_i \leq (1 + \delta) \cdot \sum_{i \in U} w_i$$

   and

$$\mathit{cost}(T') = \sum_{j \in T'} c_j \geq (1 + \delta)^{-1} \cdot \sum_{j \in T'} w_j \ .$$

In this way we obtain

$$\begin{aligned}
\mathit{cost}(U) - \mathit{cost}(T') \le & \\
& \le (1+\delta)\cdot\sum_{i\in U} w_i - (1+\delta)^{-1}\cdot\sum_{j\in T'} w_j \\
& \le (1+\delta)\cdot\sum_{i\in U} w_i - (1+\delta)^{-1}\cdot\sum_{i\in U} w_i = \frac{\delta\cdot(2+\delta)}{1+\delta}\cdot\sum_{i\in U} w_i \\
& \le \frac{\delta\cdot(2+\delta)}{1+\delta}\cdot\sum_{i\in U}(1+\delta)\cdot c_i = \delta\cdot(2+\delta)\cdot\sum_{i\in U} c_i \\
& \qquad = \delta\cdot(2+\delta)\cdot \mathit{cost}(U).
\end{aligned}$$

Finally,

$$\frac{\mathit{cost}(U) - \mathit{cost}(T')}{\mathit{cost}(U)} \le \frac{\delta\cdot(2+\delta)\cdot\mathit{cost}(U)}{\mathit{cost}(U)} = \delta\cdot(2+\delta).$$

2. Let $d = \sum_{i\in U} w_i - \sum_{j\in T'} w_j > 0$.
Let c be the cost of the first part of U with the weight $\sum_{j\in T'} w_j$. Then in the same way as in 1. one can establish

$$\frac{c - \mathit{cost}(T')}{c} \le \delta\cdot(2+\delta)\ . \tag{1}$$

It remains to bound $\mathit{cost}(U) - c$, i.e. the cost of the last part of U with the weight d. Obviously, $d \le b - \sum_{j\in T'} w_j \le w_{i_r}$ for some $r > k$, $i_r \in U$ (if not, then A_ε would add r to T' in the greedy procedure). Since $w_{i_1} \ge w_{i_2} \ge \cdots \ge w_{i_l}$

$$d \le w_{i_r} \le \frac{w_{i_1} + w_{i_2} + \cdots + w_{i_r}}{r} \le \frac{\sum_{i\in U} w_i}{k+1} \le \varepsilon\cdot\sum_{i\in U} w_i \tag{2}$$

Since $\mathit{cost}(U) \le c + d\cdot(1+\delta)$ we obtain

$$\begin{aligned}
\frac{\mathit{cost}(U) - \mathit{cost}(T')}{\mathit{cost}(U)} &\le \frac{c + d\cdot(1+\delta) - \mathit{cost}(T')}{\mathit{cost}(U)} \\
&\overset{(2)}{\le} \frac{c - \mathit{cost}(T')}{\mathit{cost}(U)} + \frac{(1+\delta)\cdot\varepsilon\cdot\sum_{i\in U} w_i}{\mathit{cost}(U)} \\
&\overset{(1)}{\le} \delta\cdot(2+\delta) + (1+\delta)\cdot\varepsilon\cdot(1+\delta) \\
&= 2\delta + \delta^2 + \varepsilon\cdot(1+\delta)^2 \\
&= \varepsilon + \delta\cdot(2+\delta)\cdot(1+\varepsilon).
\end{aligned}$$

□

We see that the PTAS SKP is stable according to $DIST$, but this does not suffice to get a PTAS for KP_δ for any $\delta > 0$. This is because in the approximation ratio we have the additive factor $\delta \cdot (2 + \delta)$ that is independent of ε. In what follows we change the PTAS SKP a little bit in such a way that we obtain a PTAS for every KP_δ, $\delta > 0$. The modification idea is very natural−to sort the input values according the cost per one weight unit.

PTAS MOD-SKP

Input: positive integers $w_1, w_2, \ldots, w_n, b, c_1, \ldots, c_n$ for some $n \in \mathbb{N}$ and some positive real number ε, $1 > \varepsilon > 0$.
Step 1: Sort $\frac{c_1}{w_1}, \frac{c_2}{w_2}, \ldots, \frac{c_n}{w_n}$. For simplicity we may assume $\frac{c_i}{w_i} \geq \frac{c_{i+1}}{w_{i+1}}$ for $i = 1, \ldots, n-1$.
Step 2: Set $k = \lceil 1/\varepsilon \rceil$.
Step 3: The same as Step 3 of PTAS SKP, but the greedy procedure follows the ordering of w_i's of Step 1.
Output: The best T' constructed in Step 3.

Let MOD-SK$_\varepsilon$ denote the algorithm given by PTAS MOD-SKP for a fixed $\varepsilon > 0$.

Lemma 5. *For every ε, $1 > \varepsilon > 0$ and every $\delta \geq 0$, MOD-SK$_\varepsilon$ is a $1+\varepsilon\cdot(1+\delta)^2$-approximation algorithm for SK$_\delta$.*

Proof. Let $U = \{i_1, i_2, \ldots, i_l\} \subseteq \{1, 2, \ldots, n\}$, where $w_{i_1} \leq w_{i_2} \leq \cdots \leq w_{i_l}$, be an optimal solution for the input $I = w_1, \ldots, w_n, b, c_1, \ldots, c_n$.

If $l \leq k$ then PTAS MOD-SKP$_\varepsilon$ provides an optimal solution.

If $l > k$, then we consider a $T' = \{i_1, i_2, \ldots, i_k, j_{k+1}, \ldots, j_{k+r}\}$ as a greedy extension of $T = \{i_1, i_2, \ldots, i_k\}$. Again, we distinguish two possibilities:

1. Let $\sum_{i \in U} w_i - \sum_{j \in T'} w_j < 0$.
   Now, we show that this contradicts to the optimality of U. Both, $cost(U)$ and $cost(T')$ contain $\sum_{s=1}^{k} c_{i_s}$. For the rest T' contains the best choice of w_i's according to the cost of one weight unit. The choice of U per one weight unit cannot be better. So, $cost(U) < cost(T')$.
2. Let $d = \sum_{i \in U} w_i - \sum_{j \in T'} w_j \geq 0$.
   Because of the optimal choice of T' according to the cost per one weight unit, the cost c of the first part of U with the weight $\sum_{j \in T'} w_j$ is at most $cost(T')$, i.e.,

$$c - cost(T') \leq 0 \ . \tag{3}$$

Since U and T' contain the same k indices $i_1, i_2, \ldots, i_k$ and $w_{i_1}, \ldots, w_{i_k}$ are the largest weights in both U and T', the same same consideration as in the proof of Lemma 1 (see (2)) yields

$$d \leq \varepsilon \cdot \sum_{i \in U} w_i, \text{ and } cost(U) \leq c + d \cdot (1 + \delta) \ . \tag{4}$$

Thus,

$$\frac{cost(U) - cost(T')}{cost(U)} \leq \frac{c + d \cdot (1+\delta) - cost(T')}{cost(U)}$$
$$\overset{(3)}{\leq} \frac{d \cdot (1+\delta)}{cost(U)} \overset{(4)}{\leq} \varepsilon \cdot (1+\delta) \cdot \frac{\sum_{i \in U} w_i}{cost(U)} = \varepsilon \cdot (1+\delta)^2 \ .$$

□

We observe that the collection of MOD-KP$_\varepsilon$ algorithms is a PTAS for every KP_δ with a constant $\delta \geq 0$ (independent of the size $2n+1$ of the input).

5 Conclusion and Discussion

In the previous sections we have introduced the concept of stability of approximations. Here we discuss the potential applicability and usefulness of this concept. Using this concept, one can establish positive results of the following types:

1. An approximation algorithm or a PTAS can be successfully used for a larger set of inputs than the set usually considered (see Lemma 1 and Lemma 4).
2. We are not able to successfully apply a given approximation algorithm A (a PTAS) for additional inputs, but one can simply modify A to get a new approximation algorithm (a new PTAS) working for a larger set of inputs than the set of inputs of A.
3. To learn that an approximation algorithm is unstable for a distance measure could lead to the development of completely new approximation algorithms that would be stable according to the considered distance measure.

The following types of negative results may be achieved:

4. The fact that an approximation algorithm is unstable according to all “reasonable” distance measures and so that its use is really restricted to the original input set.
5. Let $Q = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal) \in NPO$ be well approximable. If, for a distance measure D and a constant r, one proves the nonexistence of any polynomial-time approximation algorithm for $Q_{r,D} = (\Sigma_I, \Sigma_O, L, Ball_{r,D}(L_I), \mathcal{M}, cost, goal)$, then this means that the problem Q ist “unstable” according to D.

Thus, using the notion of stability one can search for a spectrum of the hardness of a problem according to the set of inputs. For instance, considering a hard problem like TSP one could get an infinite sequence of input languages $L_0, L_1, L_2, \ldots$ given by some distance measure, where $\varepsilon_r(n)$ is the best achievable relative error for the language L_r. Results of this kind can essentially contribute to the study of the nature of hardness of specific computing problems.

References

1. T. Andreae, H.-J. Bandelt: Performance guarantees for approximation algorithms depending on parametrized triangle inequalities. *SIAM J. Disr. Math.* 8 (1995), pp. 1–16. 41
2. S. Arora: Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In: *Proc. 37th IEEE FOCS*, IEEE 1996, pp. 2–11. 32
3. S. Arora: Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In: *Proc. 38th IEEE FOCS*, IEEE 1997, pp. 554–563. 32
4. D. P. Bovet, C. Crescenzi: *Introduction to the Theory of Complexity*, Prentice-Hall, 1993. 31, 33
5. M. A. Bender, C. Chekuri: Performance guarantees for the TSP with a parameterized triangle inequality. In: *Proc. WADS'99, LNCS*, to appear. 41
6. H.-J. Böckenhauer, J. Hromkovič, R. Klasing, S. Seibert, W. Unger: Towards the Notion of Stability of Approximation Algorithms and the Traveling Salesman Problem. Unpublished manuscript, RWTH Aachen, 1998. 41
7. N. Christofides: Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsbourgh, 1976. 31, 31
8. P. Crescenzi, V. Kann: *A compendium of NP optimization problems.* http://www.nada.kth.se/theory/compendium/. 34
9. S. A. Cook: The complexity of theorem proving procedures. In: Proc *3rd ACM STOC*, ACM 1971, pp. 151–158. 31, 31
10. M. R. Garey, D. S. Johnson: *Computers and Intractibility. A Guide to the Theory on NP-Completeness.* W. H. Freeman and Company, 1979. 33
11. K. Gödel: Über formal unentscheidbare Sätze der Principia Mathematica und verwandte Systeme, I. *Monatshefte für Mathematik und Physik* 38 (1931), pp. 173–198. 29
12. J. Håstad: Some optimal inapproximability results. In: *Proc. 29th ACM STOC*, ACM 1997, pp. 1–10.
13. D. S. Hochbaum (Ed.): *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company 1996. 33, 34, 35
14. O. H. Ibarra, C. E. Kim: Fast approximation algorithms for the knapsack and sum of subsets problem. *J. of the ACM* 22 (1975), pp. 463–468. 31, 31
15. D. S. Johnson: Approximation algorithms for combinatorial problems *JCSS* 9 (1974), pp. 256–278. 31
16. R. M. Karp: Reducibility among combinatorial problems. In: R. E. Miller, J. W. Thatcher (Eds.): *Complexity of Computer Computations*, Plenum Press 1972, pp. 85–103. 31
17. L. Lovász: On the ratio of the optimal integral and functional covers. *Discrete Mathematics* 13 (1975), pp. 383–390. 31
18. I. S. B. Mitchell: Guillotine subdivisions approximate polygonal subdivisions: Part II – a simple polynomial-time approximation scheme for geometric k-MST, TSP and related problems. Technical Report, Dept. of Applied Mathematics and Statistics, Stony Brook 1996. 32
19. E. W. Mayr, H. J. Prömel, A. Steger (Eds.): *Lecture on Proof Verification and Approximation Algorithms. Lecture Notes in Computer Science* 1967, Springer 1998. 31, 35
20. Ch. Papadimitriou: The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science* 4 (1977), pp. 237–244. 32

21. Ch. Papadimitriou: *Computational Complexity*, Addison-Wesley 1994. 31, 33
22. B. A. Trachtenbrot: *Algorithms and Automatic Computing Machines.* D. C. Heath & Co., Boston, 1963. 30
23. I. Wegener: *Theoretische Informatik: eine algorithmenorientierte Einführung*. B. G. Teubner 1993. 33

Algorithms on Compressed Strings and Arrays*

Wojciech Rytter[1,2]

[1] Department of Computer Science, Liverpool University, Chadwick Building,
Peach Street, Liverpool L69 7ZF, UK
[2] Instytut Informatyki, Uniwersytet Warszawski,
Banacha 2, 02–097 Warszawa, Poland

Abstract. We survey the complexity issues related to several algorithmic problems for *compressed* one- and two-dimensional texts *without explicit decompression*: pattern-matching, equality-testing, computation of regularities, subsegment extraction, language membership, and solvability of word equations. Our basic problem is one- and two-dimensional pattern-matching together with its variations. For some types of compression the pattern-matching problems are unfeasible (NP-hard), for other types they are solvable in polynomial time and we discuss how to reduce the degree of corresponding polynomials.

1 Introduction

In the last decade a new stream of research related to data compression has emerged: *algorithms on compressed objects*. It has been caused by the increase in the volume of data and the need to store and transmit masses of information in *compressed* form. The compressed information has to be quickly accessed and processed without explicit decompression. In this paper we consider several problems for compressed strings and arrays. The complexity of basic string problems in compressed setting for one dimensional texts is polynomial, but it jumps if we pass over to two dimensions. Our basic computational problem is the *fully compressed matching*:

Instance: $\mathcal{P} = Compress(P)$ and $\mathcal{T} = Compress(T)$, representing the compressed *pattern* and the compressed *text*.
Question: does $Decompress(\mathcal{P})$ occur in $Decompress(\mathcal{T})$?

We can change the way we formulate a problem instance by representing the pattern directly in uncompressed form , *i.e.*, as a text or an array P. This defines the *compressed matching* problem. We may also add to the problem instance the coordinates of a location of P within the text, and ask whether the text contains an occurrence of the pattern at this location. By representing the pattern in the compressed and uncompressed form we define the problems of *fully compressed pattern checking* and of *compressed pattern checking.*

In this paper we are mostly interested in *highly compressed* objects, which means that the real size (of uncompressed object) is potentially exponential with

* Supported by the grant KBN 8T11C03915.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 48–65, 1999.

respect to the size of its compressed representation. For one theoretical type of compression (in terms of morphisms) this can be even doubly exponential. In case of high compression the existence of polynomial time algorithms for many basic questions is a nontrivial challenge. High compression could be present in practical situations especially in compressing images. The theoretically interesting highly compressible strings are Fibonacci words. The n-th Fibonacci word is described by the recurrences: $F_1 := b;\ F_2 := a;\ F_n := F_{n-1} \cdot F_{n-2}$

For example

$$F_{10} = abaababaabaababaababaabaababaabaababaababaabaababaababa.$$

Two interesting examples of highly compressed arrays are the k-th rank square corner of Sierpinski triangle, denoted by S_k, see Figure 4, and the k-th order Hilbert array H_k, see Figure 3. S_k is a $2^k \times 2^k$ black and white array defined recursively: S_0 consists of a single black element, and S_k is a composition of 3 disjoint copies of S_{k-1} and totally white (blank) subarray, see Figure 1. The array H_k represents the sequence of moves in a *strongly recursive traversal* of $2^k \times 2^k$ array starting and finishing at fixed corners. The traversal of a square array is *strongly recursive* iff when entering any of its four quadrants all fields of this quadrant are visited (each one exactly once) before leaving the quadrant, and the same property holds for each quadrant and its sub-quadrants. The compressed matching problem for these particular examples is: check if an explicitly given word (array) P of total size m is a subword (subarray) of F_n $(H_n,\ S_n)$, find the number of occurrences. Observe that in both cases we cannot simply construct F_n, H_n or S_n since their uncompressed sizes are exponential with respect to n. Such description of F_k, S_k and H_h coresponds to one- and two-dimensional *straight-line programs* (SLP's), defined formally later . An SLP is a way of describing larger objects in terms of their (smaller) parts.

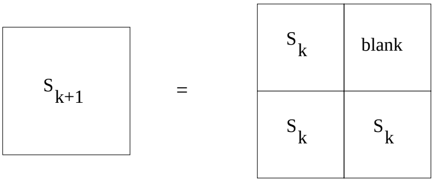


Fig. 1. The recursive structure of S_k

2 Sequential Searching in Compressed Texts

We discuss several types of 1-dimensional compression: run-length, straight-line programs (SLP's), LZ, LZW, and antidictionary compressions. We concentrate on LZ and SLP-compressions as the most interesting and closely related potentially exponential compressions. The key concepts in highly compressed matching algorithms are *periodicity* and *linearly-succinct* representations of exponentially many periods.

2.1 Periodicities in Strings

A nonnegative integer p is a *period* of a nonempty string w iff $w[i] = w[i-p]$, whenever both sides are defined.

Lemma 1. [11] *If w has two periods p, q such that $p + q \leq |w|$ then* $\gcd(p,q)$ *is a period of w, where* gcd *means "greatest common divisor".*

Denote $Periods(w) = \{p : p \text{ is a period of } w\}$. A set of integers forming an arithmetic progression is called here *linear*. We say that a set of positive integers from $[1 \ldots U]$ is *linearly-succinct* iff it can be decomposed in at most $\lfloor \log_2(U) \rfloor + 1$ linear sets. For example $Periods(aba) = \{0, 2, 3\}$.

Lemma 2. [34] *The set $Periods(w)$ is linearly-succinct w.r.t. $|w|$.*

Denote $ArithProg(i, p, k) = \{i, i+p, i+2p, \ldots, i+kp\}$, so it is an arithmetic progression of length $k+1$. Its description is given by numbers i, p, k written in binary. The size of the description, is the total number of bits in i, p, k. Denote by $Solutions(p, U, W) = (k, s)$ where k is any position $i \in U$ such that $i + j = p$ for some $j \in W$ and s is the number of such numbers k. The following lemma is used to count number of pattern occurrences in highly compressed texts.

Lemma 3. *Assume that two linear sets $U, W \subseteq [1 \ldots N]$ are given by their descriptions. Then for a given number $c \in [1 \ldots N]$ we can compute $Solutions(c, U, W)$ in polynomial time with respect to* $\log(N)$.

2.2 Run-Length Compression

The run-length compression (denoted by RLC(w)) of the string w is its representation in the form $w = a_1^{r_1} a_2^{r_2} \ldots a_k^{r_k}$, where a_i's are single symbols and $a_i \neq a_{i+1}$ for $1 \leq i < k$. Denote the size of the compressed representation by $n = |RLC(w)| = k$. We ignore here the sizes of integers and assume that each arithmetic operation takes a unit time.

Theorem 1. [3] *Assume we are given run-length encodings of the text T and the pattern P of sizes $n = |RLC(T)|$ and $m = |RLC(P)|$. Then we can check for an occurrence of P in T in $O(n+m)$ time.*

Proof. Assume that the pattern and the text are nontrivial words: each of them contains at least two distinct letters. Let $T = a_1^{r_1} a_2^{r_2} \ldots a_k^{r_k}$ and $P = b_1^{t_1} b_2^{t_2} \ldots b_s^{t_s}$. Construct

$$T' = r_2 r_3 \ldots r_{k-1}, P' = t_2 t_3 \ldots t_{s-1},$$
$$\alpha(T) = a_2 a_3 \ldots a_{k-1} \text{ and } \alpha(P) = b_2 b_3 \ldots b_{s-1}.$$

We search for all occurrences of P' in T' and simultaneously $\alpha(P)$ in $\alpha(T)$. For each starting occurrence i a constant-time additional work suffices to check if P occurs at i in T.

2.3 1-Dimensional Straight-Line Programs

A *straight-line program* (SLP) $\mathcal{R}$ is a sequence of assignment statements:

$$X_1 := expr_1;\ X_2 := expr_2; \ldots;\ X_n := expr_n$$

where X_i are variables and $expr_i$ are expressions of the form:
$expr_i$ is a symbol of a given alphabet Σ, or $expr_i = X_j \cdot X_k$, for some $j, k < i$, where $\cdot$ denotes the concatenation of X_i and X_j.

Theorem 2. [34,23] *The first occurrence and the number of all occurrences of a SLP compressed pattern in an SLP compressed text can be computed in polynomial time.*

Proof. (Sketch)
For two variables X, Y define $Overlaps(X, Y)$ as the set of all positions i in Y such that the suffix of Y which starts at i is a prefix of X. De to Lemma 2 the sets $Overlaps(X, Y)$ are lineraly-succinct. In the pattern matching problem we compute $Overlaps(X_i, P)$ and $Overlaps(P, X_j)$ for every variable X_i, X_j in the SLP describing T (bootom-up). Then we use Lemma 3 to check if there is an occurrence of the pattern overlapping the *splitting point*, see Figure 2, of some variable X_k.

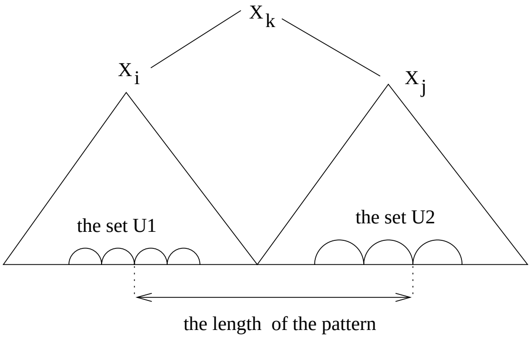


Fig. 2. A pattern occurs on a *splitting point* of the variable X_k iff $Solutions(|P|, U1, U2) \neq \emptyset$, where $U1 = Overlaps(P, X_i)$, $U2 = Overlaps(X_j, P)$

Example. The 5th Fibonacci word is described by the following *SLP*:

$$X_1 := b;\ X_2 := a;\ X_3 := X_2X_1;\ X_4 := X_3X_2;\ X_5 := X_4X_3$$

Using our algorithm it can be effectively found, for example, an occurrence (if there is any) of the Fibonacci word F_{220} in the Thue-Morse word ψ_{200} (see [40] and the last section for definition), despite the fact that *real* lengths of these strings are astronomic: $|\psi_{200}| = 2^{200}$ and $|F_{220}| \geq 2^{120}$.

Let $occ(P, T)$ be a word of length $|T|$ over the alphabet $\{0, 1\}$ such that $occ[i] = 1$ iff i is an ending position of an occurrence of P in T.

Theorem 3. *There is an SLP for $occ(P, T)$ which is of polynomial size with respect to $|SLP(T)|$ and $|P|$.*

Surprisingly the set of occurrences of a fixed pattern in a compressed text can have a very short SLP representation but its representation in terms of union of arithmetic progressions is exponential. We take the sequence of words w_i over alphabet $\{a, b\}$ such that the k-th letter of w_i is a iff the number k, written in ternary, does not contain the digit 1. (The positions in w_i are counted starting from 0.)

Theorem 4. *There are strings w_i whose SLP representation is linear but the set of all occurrences of a single letter a in w_i is not linearly succinct.*

2.4 The Lempel-Ziv Compression

The LZ compression (see [60]) gives a very natural way of representing a string and it is a practically successful method of text compression. We consider the same version of the LZ algorithm as in [18] (this is called LZ1 in [18]). Intuitively, LZ algorithm compresses the text because it is able to discover some repeated subwords. We consider here the version of LZ algorithm without *self-referencing* but our algorithms can be extended to the general self-referential case. Assume that Σ is an underlying alphabet and let w be a string over Σ. The factorization of w is given by a decomposition: $w = c_1 f_1 c_2 \ldots f_k c_{k+1}$, where $c_1 = w[1]$ and for each $1 \le i \le k$ $c_i \in \Sigma$ and f_i is the longest prefix of $f_i c_{i+1} \ldots f_k c_{k+1}$ which appears in $c_1 f_1 c_2 \ldots f_{i-1} c_i$. We can identify each f_i with an interval $[p, q]$, such that $f_i = w[p..q]$ and $q \le |c_1 f_1 c_2 \ldots f_{i-1} c_{i-1}|$. If we drop the assumption related to the last inequality then a *self-referencing* occurs (f_i is the longest prefix which appears before but not necessarily terminates at a current position). We assume that this is not the case.

Example. The factorization of a word $aababbabbaababbabba\#$ is given by:

$$c_1\ f_1\ c_2\ f_2\ c_3\ f_3\ c_4\ f_4\ c_5 = a\ a\ b\ ab\ b\ abb\ a\ ababbabba\ \#.$$

After identifying each subword f_i with its corresponding interval we obtain the LZ encoding of the string. Hence

$$LZ(aababbabbababbabb\#) = a[1,1]b[1,2]b[4,6]a[2,10]\#.$$

Lemma 4. *For each string w given in LZ-compressed form we can construct an SLP generating w of size $O(n^2)$, where $n = |LZ(w)|$.*

Theorem 5. [18] *The compressed pattern-matching for LZ-encoded texts can be done in $O(n \cdot \log^2(|T|/n) + |P|)$ time.*

When we pass to the fully compressed pattern-matching the situation is more complicated. We can use Lemma 4 to reduce the problem to the SLP-compressed matching. Another approach was used in [23], where a generalization of SLP's has been introduced and applied: a *composition system*. Denote by $Eq(n)$ the time to check subwords-equality of LZ-compressed texts. The technique of randomized *fingerprinting* has been used to show:

Theorem 6. [24] *Assume a string w is given in LZ-compressed form, then we can preprocess w in $O(n^2 \log n)$ time in such a way that each subword equality query about w can be answered in $O(n{\cdot}\log\log n)$ time with a very small probability of error.*

Theorem 7. [23] *The Fully Compressed Matching Problem for LZ-compressed strings can be solved in $O(n \log n \log^2 |T| \cdot Eq(n \log n))$ time.*

Theorem 8. *The linearly-succinct representation of the set Periods$(\mathcal{P})$ can be computed in $O(n \log n \log^2 |T| \cdot Eq(n \log n))$ time.*

Theorem 9. [23] *We can test if an LZ-compressed text contains a palindrome in $O(n \log n \log^2 |T| \cdot Eq(n \log n))$ time.*
We can test for square-freeness in $O(n^2 \log^2 n \log^3 |T| \cdot Eq(n \log n))$ time.

Theorem 10. [23] *Given text T, its code $LZ(T)$ can be computed on-line with $O(n^2 \log^2 n)$ delay using $O(n \log n)$ space.*

2.5 LZW Compression

The main difference between LZ and LZW compression is in the choice of *code words*. Each next code word is of a form wa, where w is an earlier code-word and a is a letter. The text is scanned left to write, each time the next largest code-word is constructed. The code-words form a trie, and the text to be compressed is encoded as a sequence of names of prefixes of the trie. Denote by $LZW(w)$ the Lempel-Ziv-Welch compression of w. This type of compression cannot compress the string exponentially, it is less interesting from the theoretical point of view but much more interesting from the practical point of view.

Lemma 5. $|LZW(w)| = \Omega(\sqrt{|w|})$.

Theorem 11. [2] *The compressed matching problem for LZW-encoded strings can be done in $O(\min\{n + m^2,\ n \cdot \log(m) + m\})$ time.*

Theorem 12. [25] *The fully compressed matching problem for LZW-encoded strings can be done in $O((n + m) \cdot \log(n + m))$ time.*

2.6 Texts Compressed by Using Antidictionaries

The method of data compression that uses some “negative” infornmation about the text is quite different from the other compression method. Assume in this subsection that the alphabet is binary. Let $AD(w)$ denote the set (called the *antidictionary*) of minimal *forbidden* factors of w, this means words which are not subwords of w. The minimality is in the sense of subword inclusion.

For example $AD(01001010) = \{000,\ 10101,\ 11\}$.
$AD(11010001) = \{0000,\ 111,\ 011,\ 0101,\ 1100\}$
The compression method processes w from left-to right symbol by symbol and produces the compressed representation $o_1 o_2 \dots o_N$, where each o_i is a single letter or an empty string. Assume that we scan the i-th letter a of w, if there is a word $ub \in AD(w)$ such that u is a suffix of $w[1 \dots i-1]$ then the i-th output word o_i is the empty word, otherwise it is a. In other words, if the i-th letter is predictable from $w[1 \dots i-1]$ and $AD(w)$, we can skip the i-th letter, since it is *"redundant"*.

Denote $ADC(w) = (AD(w), o_1 o_2 \dots o_N, N)$, where $N = |w|$.
$ADC(w)$ is the *antidictionary compressed representation* of w.

Let $|AD(w)|$ denote the total length of all strings in $AD(w)$ and the size of compressed representation is defined as:

$$|ADC(w)| = |AD(w)| + |o_1 o_2 \dots o_N| + \log N.$$

Observe that N is an important part of the information necessary to identify w. For example

$$ADC(1^{1000}) = (\{0\}, \epsilon, 1000),\ ADC(1^{10}) = (\{0\}, \epsilon, 10).$$

This is a trivial example of an exponential compression. It can happen that $|ADC(w)| > |w|$, for example

$$ADC(11010001) = (\{0000,\ 111,\ 011,\ 0101,\ 1100\}, 110, 8)$$

Theorem 13. [10] *$ADC(w)$ can be computed in time $O(n + N)$, where $n = |ADC(w)|$, $N = |w|$.*

Theorem 14. [57] *All occurrences of a pattern in w can be found in time $O(n + |P|^2 + r)$, where $n = |ADC(w)|$, and r is the number of occurrences of the pattern.*

3 Sequential Searching in Compressed Arrays

3.1 2-Dimensional Run-Length Encoding

For a 2-dimensional array T denote by $2RLC(T)$ the concatenation of run-length encodings of the rows of T.

Theorem 15. [4] *Assume we are given run-length encoding of a 2D-text T and an explicitly given 2D-pattern P, where $n = |2RLC(T)$ and $M = |(P)|$. Then we can check for an occurrence of P in T in $O(n + M)$ time.*

3.2 2-Dimensional Straight-Line Programs

A 2D-text can be represented by a 2-dimensional straight-line program (SLP), that uses constants from the alphabet, and two types of assignment statements

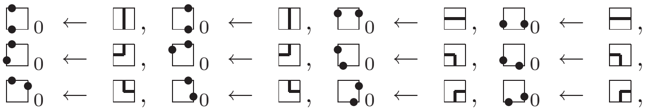
Horizontal concatenation: $A \leftarrow B \oslash C$, which concatenates 2D-texts B and C (both of equal height)

Vertical concatenation: $A \leftarrow B \ominus C$, which puts the 2D-text B on top of C (both of equal width)

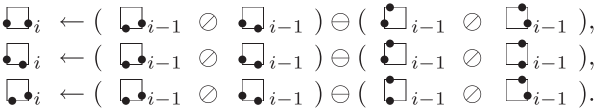
An SLP $\mathcal{P}$ of size n (we write $|\mathcal{P}| > n$) consists of n statements of the above form, where the result of the last statement is the compressed 2D-text. The result of a correct SLP, $\mathcal{P}$, is denoted *Decompress*($\mathcal{P}$). We say that $\mathcal{P}$ is a *compressed form* of P. The *area* of $P =$ *Decompress*($\mathcal{P}$), denoted $|P|$, can be exponential in $|\mathcal{P}|$.

Example. Hilbert's curve can be viewed as an image which is exponentially compressible in terms of SLP's. An SLP which describes the n^{th} Hilbert's curve, H_n, uses six (terminal) symbols □, □, □, □, □, □, and 12 variables $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, $\square_i$, for each $0 \leq i \leq n$. A variable with index i represents a text square of size $2^i \times 2^i$ containing part of a curve. The dots in the boxes show the places where the curve enters and leaves the box.

The 2D-text $T = \square_3$ describing the 3rd Hilbert's curve is shown in Figure 3. It is composed of four smaller square 2D-texts $\square_2$, $\square_2$, $\square_2$ and $\square_2$ according to one of the composition rules. In the figure the black dots indicate how T was defined with statement $\square_3 \leftarrow (\ \square_2 \oslash \square_2\ ) \ominus (\ \square_2 \oslash \square_2\ )$ The 1×1 text squares are described as follows.



The text squares for variables indexed by $i \geq 1$ are rotations of text squares for the variables $\square_i$, $\square_i$, $\square_i$. These variables are composed according to the rules:



Theorem 16. [9]

(1) *There exists a polynomial time randomized algorithm for testing equality of two 2D-texts, given their SLPs.*

(2) *Fully compressed pattern checking for 2D-texts is co-***NP***-complete.*

(3) *Fully compressed matching for 2D-texts is* $\Sigma_2^{\mathbf{P}}$*-complete.*

(4) *Compressed matching for 2D-texts is* **NP***-complete.*

3.3 2D-Compressions in Terms of Finite Automata

Finite automata can describe quite complicated images, for example deterministic automata can describe the Hilbert's curve with a given resolution, see Figure 3, while weighted automata can describe even much more complicated curves,

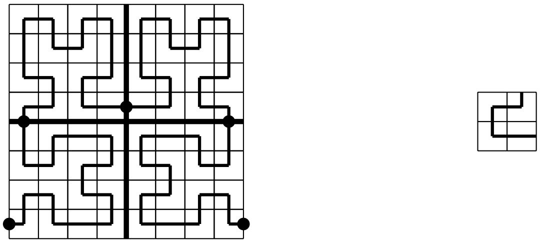


Fig. 3. An example of a 2D-text T and a pattern P. The pattern occurs twice in the text

see also [13,14,16]. The automata can be also used as an effective tool to compress two-dimensional images, see [35,15].

Our alphabet is $\Sigma = \{0,\ 1,\ 2,\ 3\}$, the elements of which correspond to four quadrants of a square array, listed in a fixed order. A word w of length k over Σ can be interpreted, in a natural way, as a unique address of a pixel x of a $2^k \times 2^k$ image (array), we write $address(x) = w$. The length k is called the *resolution* of the image. For a language $L \subseteq \Sigma^+$ denote by $Image_k(L)$ the $2^k \times 2^k$ black-and-white image such that the color of a given pixel x is black iff $address(x) \in L$. Formally, the weighted language is a function which associates with each word w a value $weight_L(w)$. A weighted language L over Σ and resolution k determine the gray-tone image $Image_k(L)$ such that the color of a given pixel x equals $weight_L(address(x))$. We define

$$Image_k(A) = Image_k(L(A))$$

where $L(A)$ is the language accepted by A.

Example.
$Image(\Sigma^{k-1}(0 \cup 3))$ is the $2^k \times 2^k$ black-and-white chess-board.

Weighted finite automata and deterministic automata correspond to images of infinite resolution. Since we consider images of a given finite resolution k we can assume that the considered automata are acyclic.

Theorem 17. [20]
(1) *Compressed matching for deterministic automata can be solved in polynomial time.*
(2) *Compressed matching for weighted automata is NP-complete.*

3.4 2D-Compression Using LZ-Encodings

A natural approach to (potentially exponential) compression of images is to scan a given two-dimensional array T in some specified order, obtain a linear version of T called *linear*(T), and then apply Lempel-Ziv encoding to the string *linear*(T).

The *Hilbert's curve* H_k corresponds to a *strongly regular* traversal of $2^k \times 2^k$ grid, starting in a bottom left corner of $n \times n$ array T, and ending at the right bottom corner, where $n = 2^k$. An example of H_3 is illustrated in Figure 3.

We denote now by *H-linear*(T) the linearization of T according to the Hilbert's curve. The *2LZ-compression* is defined as follows:

$$2LZ(T) = LZ(H\text{-}linear(T)).$$

Such type of encoding was already considered in [39]. $2LZ$-compression is as strong as *finite automata* compression, with respect to polynomial reduction.

Theorem 18. *If A describes an image T then $2LZ(T)$ is of polynomial size w.r.t. A.*

Theorem 19. *Searching for an occurrence of a row of ones in a $2LZ$ compressed image is NP-hard.*

Surprisingly there are black-and-white images whose $2LZ$ encoding is small and any *deterministic acyclic finite automata* encoding should be exponential.

Theorem 20. *For each m there is an image W_m such that $2LZ(W_m)$ is of size $O(m)$ but each deterministic automaton encoding the image W_m contains at least 4^{m-1} states.*

4 Compressibility of Subsegments

In this section we consider the problem of constructing a compressed representation of a part of a compressed object. The compressed representation of a part can be larger than that of the whole object. We start with finite-automata representations. In our considerations we may restrict to acyclic automata since we consider only finite resolution images.

Example. *Image*$(\{0, 1, 2\}^k) = S_k$ is the $2^k \times 2^k$ black-and-white square part of Sierpinski's triangle. see Figure 4 for the case $k = 4$. The corresponding smallest acyclic deterministic automaton accepting all paths describing black pixels has 5 essential states but there are needed 9 essential states to describe the 8×8 subarray R of S_4 indicated in Figure 4. (The state is *essential* iff it is on an accepting path, other states are treated as redundant.)

Theorem 21. [9] *Assume the compression is in terms of deterministic automata. Let n be the size of a deterministic automaton describing $\mathcal{T}$.*
(a) *The compressed representation of a subsquare R of T can be computed in $O(n^{2.5})$ time.*
(b) *For each subimage $\mathcal{R}$ of an image $\mathcal{T}$ there is a deterministic automaton describing $\mathcal{R}$ of size $O(n^{2.5})$. There are images $\mathcal{T}$ and their subimages $\mathcal{R}$ such that the smallest deterministic automaton for $\mathcal{R}$ requires $\Omega(n^{2.5})$ states.*

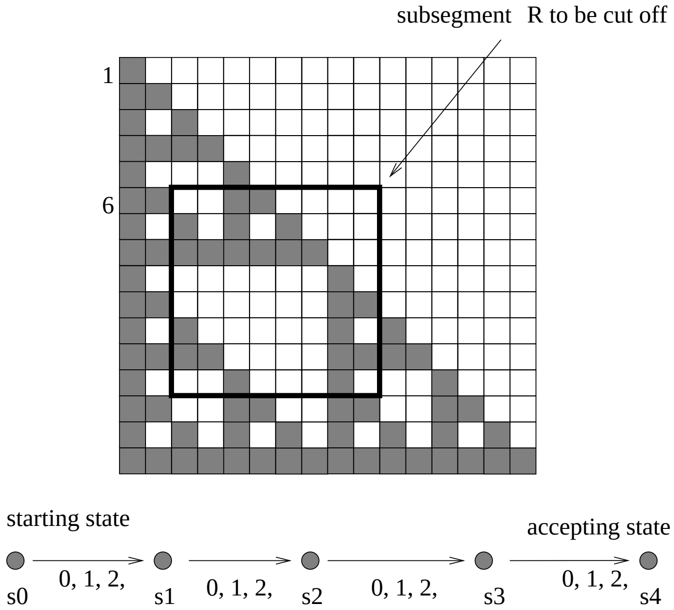


Fig. 4. The image S_4 and its smallest acyclic automaton. Edges which are not on an accepting path are disregarded

The situation is much different for 2-dimensional straight-line programs.

Theorem 22. [29] *For each n there exists an* SLP *of size n describing a text image A_n and a subrectangle B_n of A_n such that the smallest* SLP *describing B_n has exponential size.*

5 Parallel Searching in Compressed Texts and Arrays

The difference between sequential and NC-computations for compressed texts can be well demonstrated by the following problem: compute the symbol $T[i]$ where T is the text given in its LZ-version, and i is given in binary.

This task has a trivial sequential linear time algorithm (which performs computations sequentially in the order the recurrences are written). However if we ask for an NC-algorithm for the same problem the situation is different, and it becomes quite difficult. Any straightforward attempt of using the doubling technique and squaring the matrix of positions fails, since the number of positions in the text defined by n recurrence equations can be $\Omega(2^n)$.

Theorem 23. [26]
(1) *The problem of testing for any occurrence of an uncompressed pattern in a LZ-compressed text is P-complete.*

(2) *The problem of computing a symbol on a given position in a LZ-compressed text is P-complete.*

Theorem 24. [26] *The SLP-compressed matching problem can be solved in* $O(\log(m) \cdot \log(n))$ *time with* $O(n)$ *processors.*

Theorem 25. [26] *The pattern-matching problem for SLP-compressed 2d-texts can be solved in*
(1) $O(\log^2(n+m))$ *time with* $O(n^2 \cdot m + n^6)$ *processors, or*
(2) $O(n + \log m)$ *time with* $O(n \cdot (n+m))$ *processors.*

Theorem 26. [25] *There is an almost optimal NC algorithm for fully compressed LZW-matching.*

Theorem 27. [6,12]
The computation of $LZW(w)$ *is P-complete.*
The computation of $LZ(w)$ *is in* NC*.*

6 The Compressed Language Membership Problems

The language membership problem is to check if $w \in L$, given $Compress(w)$ and a description of a formal language L.

Theorem 28.
(a) *We can test the membership problem for LZ-compressed words in a language described by given regular expression* W *in* $O(n \cdot m^3)$ *time, where* $m = |W|$*.*
(b) *We can decide for LZ-compressed words the membership in a language described by given determiniatic automaton* M *in* $O(n \cdot m)$ *time, where* m *is the number of states of* M*.*

We use the following problem to show NP-hardness of several compressed recognition problems.

SUBSET SUM problem:
Input instance: Finite set $A = \{a_1, a_2, \ldots, a_n\}$ of integers and an integer K. The size of the input is the number of bits needed for the binary representation of numbers in A and K.
Question: Is there a subset $A' \subseteq A$ such that the sum of the elements in A' is exactly K?

Lemma 6. *The problem SUBSET SUM is NP-complete.*

Proof. See [31], and [22], pp. 223.

Theorem 29. [34,53] *Testing the membership of a compressed unary word in a language described by a* star-free *regular expression with compressed constants is NP-complete.*

It is not obvious if the previously considered problem is in NP for regular expressions containing the operation *, in this case there is no polynomial bound on the length of accepting paths of A. There is a simple argument in case of unary languages. In the proof of the next theorem an interesting application of the Euler path technique to a unary language recognition is shown.

Theorem 30. [53]
(a) *The problem of checking membership of a compressed unary word in a language described by a given regular expression with compressed constants is in NP.*
(b) *The problem of checking membership of a compressed word in a language described by a semi-extended regular expression is NP-hard.*

Theorem 31. *The problem of checking membership of a compressed word in a given linear context-free language L is NP-hard, even if L is given by a context-free grammar of a constant size.*

Proof. Take an instance of the subset-sum problem with the set $A = \{a_1, a_2, \ldots, a_n\}$ of integers and an integer K. Define the following language:

$$L = \Big\{ d^R \$ d^{v_1} \# d^{v_2} \# \ldots \# d^{v_t} \,:\, t \geq 1 \text{ and there is a subset } A' \subseteq \{v_1, \ldots, v_t\} \text{ such that } \sum_{u \in A'} u \;=\; R \Big\}?$$

L is obviously a linear context-free language generated by a linear context-free grammar of a constant size. We can reduce an instance of the subset sum-problem to the membership problem:

$$d^K \$ d^{a_1} \# d^{a_2} \# \ldots \# d^{a_n} \;\in\; L.$$

Theorem 32.
(a) *The problem of checking membership of a compressed word in a given linear cfl is in $NSPACE(n)$.*
(b) *The problem of checking membership of a compressed word in a given cfl is in $DSPACE(n^2)$.*

Proof. We can easily compute in linear space a symbol on a given position in a compressed input word. Now we can use a space-efficient algorithm for the recognition of context-free languages. It is known that linear languages can be recognized in $O(logN)$ nondeterministic space and general cfls can be done in $O(log^2 N)$ deterministic space, where N is the size of the uncompressed input word. In our case $N = O(2^n)$, this gives required $NSPACE(n)$ and $DSPACE(n^2)$ complexities.

Theorem 33. *The problem of checking membership of a compressed unary word in a given cfl is NP-complete.*

7 Word Equations

Word equations are used to describe properties and relations of words, e.g. pattern-matching with variables, imprimitiveness, periodicity, and conjugation, see [30]. The main algorithm in this area was Makanin's algorithm for solving word equations, see [41]. The time complexity of the algorithm is too high, and the algorithm is too complicated. Recently much simpler algorithms were constructed by W. Plandowski [51,52].

Let Σ be an alphabet of constants and Θ be an alphabet of variables. We assume that these alphabets are disjoint. A word equation E is a pair of words $(u, v) \in (\Sigma \cup \Theta)^* \times (\Sigma \cup \Theta)^*$ usually denoted by $u = v$. The *size* of an equation is the sum of lengths of u and v. A *solution* of a word equation $u = v$ is a morphism $h : (\Sigma \cup \Theta)^* \to \Sigma^*$ such that $h(a) = a$, for $a \in \Sigma$, and $h(u) = h(v)$. For example assume we have the equation

$$abx_1x_2x_2x_3x_3x_4x_4x_5 = x_1x_2x_3x_4x_5x_6,$$

and the length of x_i's are consecutive Fibonacci numbers. Then the solution $h(x_i)$ is the i-th Fibbonaci word.

It is known that the solvability problem for word equations is NP-hard, even if we consider (short) solutions with the length bounded by a linear function and the right side of the equation contains no variables, see [5].

The main open problem is to show the following:

Conjecture A: The problem of solving word equations is in NP.

Conjecture B: Let $\mathcal{N}$ be the minimal length of the solution (if one exists). Then $\mathcal{N}$ is singly exponential w.r.t. n.

The author believs that both questions have positiove answers.

A motivation to consider compressed solutions follows from the following fact (which is an application of a fully compressed pattern-checking, in this case checking occurrence of one compressed string at the beginning of another one).

Lemma 7. *If we have LZ-encoded values of the variables then we can verify the word equation in polynomial time with respect to the size of the equation and the total size of given LZ-encodings.*

Theorem 34. *Assume N is the size of minimal solution of a word equation of size n. Then each solution of size N can be LZ-compressed to a string of size $O(n^2 \log^2(N)(\log n + \log\log N))$.*

As a direct consequence of Lemma 7 and Theorem 34 we have:

Conjecture B implies conjecture A.

Theorem 35. *Assume the length of all variables are given in binary by a function f. Then we can test solvability in deterministic polynomial time, and produce polynomial-size LZ-compression of the lexicographically first solution (if there is any).*

8 Morphic Representations

There is a classical (in terms of formal language theory) short description of strings in terms of morphisms. Assume $w = \phi^k(a)$, where ϕ is a morphism and a is a letter in the alphabet. Words of the type $\phi^k(a)$ can be interpreted as instructions in a "turtle language" to draw fractal images, see [49]. Hence the calculation of the i-th letter has a practical meaning when computing a local structure of a fractal without computing the whole object. The pair (k, ϕ) can be treated as short morphic description of w, denoted by $morphic\_desc(w)$. The length $n = |morphic\_desc(w)|$ of the description is the size of the binary representation of k and ϕ.

For some morphisms the computation of the i-th letter could be especially simple. This is the case for the Thue-Morse morphism

$$\Psi(0) = 01,\ \Psi(1) = 10$$

Assume we count positions starting from 0. Let $bin(i)$ be the binary representation of i. Then:

$$\psi^k(0) = 1 \Leftrightarrow bin(i) \text{ contains an odd number of ones}$$

For example the 1024-th position of $\psi^{100}(0)$ is 1 since $bin(1024) = 10000000000$ contains odd number of ones. However such simple computation of the i-th letter does not apply to every morphism.

Theorem 36. *Let ϕ be a morphism. There is a polynomial time algorithm to compute the i-th letter of $\phi^k(a)$, where the input size is the total number of bits for i, ϕ, and k.*

Instead of morphic functions ϕ we can use a finite-state transducer function λ_A where A is a finite state transducer (deterministic finite automaton with an output function). The replacement of ϕ by λ_A has dramatic consequences.

Theorem 37. *[56] Let A be a finite state transducer. The problem of computing the i-th letter of $\lambda_A^k(a)$ is EXPTIME-hard.*

9 Final Remarks

We have surveyed complexity-theoretical results related to the processing of large compressed texts and arrays without decompression. However we have not discussed one important aspects: practicality of algorithms. Recently many related practical issues were investigated, see [32,33,45,46,47]. The whole area is in an initial stage.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The design and analysis of computer algorithms*, *Addison-Wesley*, Reading, Mass., 1974.
2. A. Amir, G. Benson and M. Farach, Let sleeping files lie: pattern-matching in Z-compressed files. *Journal of Computer and System Sciences*, 1996, Vol. 52, No. 2, pp. 299–307. 53
3. A. Amir, G. Benson, Efficient two dimensional compressed matching, *Proceedings of the* 2nd *IEEE Data Compression Conference*, pp. 279–288 (1992). 50
4. A. Amir, G. Benson and M. Farach, Optimal two-dimensional compressed matching, *Journal of Algorithms*, 24(2):354–379, August 1997. 54
5. Angluin D., Finding patterns common to a set of strings, *J.C.S.S.*, 21(1), 46–62, 1980. 61
6. S. De Agostino, P-complete problems in data compression, *Theoretical Computer Science* 127, 181–186, 1994. 59
7. S. De Agostino, Pattern-matching in text compressed with the ID heuristic, in J. Atorer, and M. Cohn (Editors), *Data Compression Conference* 1998, IEEE Computer Society, pp. 113–118.
8. M. F. Barnsley, L. P. Hurd, Fractal image compression, A. K. Peters Ltd. 1993.
9. P. Berman, M. Karpinski, L. L. Larmore, W. Plandowski, and W. W. Rytter, On the Complexity of Pattern Matching for Highly Compressed Two-Dimensional Texts, *Proceedings of the* 8th *Annual Symposium on Combinatorial Pattern Matching* LNCS 1264, Edited by A. Apostolico and J. Hein, (1997), pp. 40—51. 55, 57
10. M. Crochemore, F. Mignosi, A. Restivo, S. Salemi, Text compression using antidictionaries, *ICALP* 1999. 54
11. M. Crochemore and W. Rytter, *Text Algorithms,* Oxford University Press, New York (1994). 50
12. M. Crochemore, W. Rytter, Efficient parallel algorithms to test square-freeness and factorize strings, Information Processing Letters, 38 (1991) 57–60. 59
13. K. Culik and J. Karhumäki, Finite automata computing real functions, *SIAM J. Comp* (1994). 56
14. K. Culik and J. Kari, Image compression using weighted finite automata, *Computer and Graphics* 17, 305–313 (1993). 56
15. K. Culik and J. Kari, *Fractal image compression: theory and applications*, (Ed. Y. Fisher), Springer Verlag 243–258 (1995). 56
16. D. Derencourt, J. Karhumäki, M. Letteux and A. Terlutte, On continuous functions computed by real functions, *RAIRO Theor. Inform. Appl.* 28, 387–404 (1994). 56
17. S. Eilenberg, *Automata, Languages and Machines*, Vol.A, Academic Press, New York (1974).
18. M. Farach and M. Thorup, String matching in Lempel-Ziv compressed strings, *Proceedings of the* 27th *Annual Symposium on the Theory of Computing* (1995), pp. 703–712. 52, 52, 52
19. M. Farach, S. Muthukrishnan, Optimal Parallel Dictionary Matching and Compression *SPAA* 1995.
20. J. Karhumaki, W. Plandowski, W. Rytter, Pattern matching for images generated by finite automata, *FCT'97*, in LNCS Springer Verlag 1997. 56
21. M. Gu, M. Farach, R. Beigel, An Efficient Algorithm for Dynamic Text Indexing (SODA '94).

22. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* Freeman, New York (1979). 59
23. L. Gąsieniec, M. Karpinski, W. Plandowski and W. Rytter, Efficient Algorithms for Lempel-Ziv Encoding, *Proceedings of the* 5th *Scandinavian Workshop on Algorithm Theory*. Springer-Verlag (1996). 51, 52, 53, 53, 53
24. L. Gasieniec, M. Karpinksi, W. Plandowski, W. Rytter, Randomized algorithms for compressed texts: the finger-print approach, *Combinatorial Pattern Matching* 1996, Lecture Notes in Comp. Science, Springer Verlag 1996. 53
25. L. Gasieniec, W. Rytter, Almost optimal fully compressed LZW-matching, in *Data Compression Conference*, IEEE Computer Society 1999. 53, 59
26. L. Gasieniec, A. Gibbons, W. Rytter, The parallel complexity of pattern-searching in highly compressed texts, in *MFCS* 1999. 58, 59, 59
27. A. Gibbons, W. Rytter, Efficient parallel algorithms, Cambridge University Press 1988.
28. O. H. Ibarra and S. Moran, Probabilistic algorithms for deciding equivalence of straight-line programs, *JACM* 30 (1983), pp. 217–228.
29. J. Karhumaki, W. Plandowski, W. Rytter, The compression of subsegments of compressed images, in *Combinatorial Pattern Matching* 1999. 58
30. Karhumäki J., Mignosi F., Plandowski W., The expressibility of languages and relations by word equations, *in ICALP'97*, LNCS 1256, 98–109, 1997. 61
31. R. M. Karp, Reducibility among combinatorial problems, in *"Complexity of Computer Computations", Plemum Press, New York*, 1972 (Editors R. E. Miller and J. W. Thatcher). 59
32. T. Kida, M. Takeda, A. Shinohara, S. Arikawa, Sfift-and approach to pattern-matching in LZW compressed text, *Combinatorial Pattern-Matching* 1999, LNCS 1645, pp. 1–13. 62
33. T. Kida, M. Takeda, A. Shinohara, M. Miyazaki, S. Arikawa, Multiple pattern-matching in LZW compressed text, in J. Atorer, and M. Cohn (Editors), *Data Compression Conference* 1998, IEEE Computer Society, pp. 103–112. 62
34. M. Karpinski, W. Rytter and A. Shinohara, Pattern-matching for strings with short description, *Nordic Journal of Computing*, 4(2):172–186, 1997. 50, 51, 60
35. J. Kari, P. Franti, Arithmetic coding of weighted finite automata, *RAIRO Theor. Inform. Appl.* 28 343–360 (1994). 56
36. R. M. Karp and M. Rabin, *Efficient randomized pattern matching algorithms,* IBM Journal of Research and Dev. 31, pp. 249–260 (1987).
37. D. Knuth, *The Art of Computing, Vol. II: Seminumerical Algorithms. Second edition.* Addison-Wesley (1981).
38. A. Lempel and J. Ziv, On the complexity of finite sequences, *IEEE Trans. on Inf. Theory* 22 (1976) pp. 75–81.
39. A. Lempel and J. Ziv, Compression of two-dimensional images sequences, *Combinatorial algorithms on words* (Ed. A. Apostolico, Z. Galil), Springer-Verlag (1985) pp. 141–156. 57
40. M. Lothaire, *Combinatorics on Words.* Addison-Wesley (1993). 51
41. Makanin, G. S., The problem of solvability of equations in a free semigroup, *Mat. Sb.*, Vol. 103,(145), 147–233, 1977. English transl. in *Math. U.S.S.R.* Sb. Vol 32, 1977. 61
42. U. Manber, A text compression scheme that allows fast searching directly in the compressed file, *ACM Transactions on Information Systems*, 15(2), pp. 124–136, 1997.

43. M. Miyazaki, A. Shinohara, M. Takeda, An improved pattern-matching algorithm for strings in terms of straight-line programs, *Combinatorial Pattern-Matching* 1997, LNCS 1264, pp. 1–11.
44. R. Motwani, P. Raghavan, *Randomized Algorithms*, Cambridge University Press (1995).
45. E. de Moura, G, Navarro, N. Ziviani, R. Baeza-Yates, Direct pattern-matching on compressed text, in SPIRE, pp. 90–95, IEEE CS Press, 1998. 62
46. E. de Moura, G, Navarro, N. Ziviani, R. Baeza-Yates, Fast sequential searching on compressed texts allowing errors, *21st Annual Int. ACM SIGIR Conference on Research and Development in Information retrieval*, pp. 298–306, York Press 1998. 62
47. G. Navarro, M. Raffinot, A general practical approach to pattern-matching over Zil-Lempel compressed text, *Combinatorial Pattern-Matching* 1999, LNCS 1645, pp. 14–36. 62
48. Ch. H. Papadimitriou, *Computational Complexity*, Addison Wesley (1994).
49. H. O. Peitgen, P. H. Richter, The beauty of plants, Springer-Verlag 1986. 62
50. W. Plandowski, Testing equivalence of morphisms on context-free languages, *Proceedings of the* 2^{nd} *Annual European Symposium on Algorithms* (ESA'94), LNCS 855, Springer-Verlag (1994), pp. 460–470.
51. W. Plandowski, Solvability of word equations with constants is in NEXPTIME, *STOC* 1999. 61
52. W. Plandowski, Solvability of word equations with constants is in P-SPACE, *FOCS* 1999. 61
53. W. Plandowski, W. Rytter, Complexity of compressed recognition of formal languages, in *"Jewels forever"*, Springer Verlag 1999 (Ed. J. Karhumaki). 60, 60
54. W. Plandowski, W. Rytter, Applying Lempel-Ziv encodings to the solution of word equations, *ICALP* 1998.
55. J. Schwartz, Fast probabilistic algorithms for verification of polynomial identities, *J. ACM* 27 (1980) pp. 701–717.
56. J. Shallit, D. Swart, An efficient algorithm for computing the i-th letter of $\phi^n(a)$, *SODA* 1999. 62
57. Y. Shibata, M. Takeda, A. Shinohara, S. Arikawa, Pattern-matching in text compressed by using antidictionaries, *Combinatorial Pattern-Matching* 1999, LNCS 1645, pp. 37. 54
58. J. Storer, *Data compression: methods and theory,* Computer Science Press (1988).
59. R. E. Zippel, Probabilistic algorithms for sparse polynomials, *Proceedings of the International Symposium on Symbolic and Algebraic Manipulation* (EUROSAM '79) LNCS 72, Springer-Verlag (1979), pp. 216–226.
60. J. Ziv and A. Lempel, A Universal algorithm for sequential data compression, *IEEE Transactions on Information Theory* IT–23 (1977), pp. 337–343. 52

WWW Based Collaboration with the BSCW System

Wolfgang Appelt

GMD – German National Research Center for Information Technology
Schloß Birlinghoven, D-53754 Sankt Augustin, Germany
appelt@gmd.de

Abstract. The World Wide Web offers a great deal of potential in supporting cross-platform cooperative work within locally dispersed working groups. GMD has developed the BSCW System, a Web based groupware tool using the metaphor of shared virtual workspaces. The system is particularly useful – and already used by a large community – for cooperation between researchers in distributed environments. This paper describes the principles, architecture and functionality of the current (August 1999) version.

1 Introduction

Collaboration between researchers involves a rich set of modes and means of cooperation. For example, several researchers may meet spontaneously, e.g., at a conference, and discuss new research ideas. They may decide to write a joint paper, distribute the off-line drafting of different sections of the paper to individuals, have face to face meetings to discuss the drafts, maybe with mutual reviews between meetings, until the final paper eventually emerges and is presented to the scientific community. Depending on the area of research, besides textual communication additional media such as graphics, spreadsheets, animations, presentation of software or the results of experiments will be involved in their cooperation.

To enable efficient ways of cooperation, these collaboration processes need to be supported by electronic means, in particular, when cooperation takes place within locally dispersed groups. These electronic cooperation tools need to support the usual work practices of researchers, in particular, they need to provide

- a rich variety of tools for asynchronous and synchronous collaboration,
- a smooth transition between asynchronous and synchronous modes of collaboration,
- a close integration into the normal working environments of the users, and
- cross-platform interoperability, since in general cross-organisational research groups use a variety of platforms.

In the last years the Internet and the World Wide Web (WWW) in particular have become the most important infrastructure for communication within the

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 66–78, 1999.

research community. Email over the Internet has emerged as the primary means of interchanging multimedia information between researchers, and the WWW has become an important medium for dissemination of research results. The WWW has a number of advantages as the basis for tools to support collaborative information sharing:

- WWW browsers are available for all important platforms and provide access to information in a platform independent manner.
- Browsers offer a simple and consistent user interface across different platforms.
- Browsers are already part of the computing environment in many organisations.
- Many organisations have also installed their own Web servers and are familiar with server maintenance.

Given these characteristics, the extension of the Web to provide richer forms of cooperation support for working groups is both appropriate and desirable. Therefore, the CSCW (Computer Supported Cooperative Work) research group in GMD's Institute for Applied Information Technology (FIT) has developed the BSCW (Basic Support for Cooperative Work) system within the last five years which as its main goal seeks to transform the Web from a primarily passive information repository to an active cooperation medium.

2 General Approach of the BSCW System

Over the last years, CSCW research has led to a better understanding how to support electronic cooperation within groups in various environments. Empirical studies have shown (see e.g. [3]) the importance of joint information spaces (often called *shared workspaces*) particularly in locally distributed, loosely organised groups. The groups use such workspaces for the collection and structuring of any kind of information they need (e.g., documents, graphics, spreadsheets, tables, or software) to achieve the goals of their collaboration.

Such workspaces support primarily asynchronous modes of communication. This mode is normally the most important one for cooperation between researchers since in such an environment cooperation consists often in parallel, loosely coupled activities of the individual group members. Synchronous types of cooperation such as audio/video conferencing or chat sessions are usually of less importance but should also be supported to some extent. The usage of workflow systems – which are primarily addressing the execution of a set of tasks following a predefined sequence with allocation of responsibilities to persons or roles – is normally not appropriate in such groups.

The BSCW system is based on the metaphor of *shared workspaces*. The users access these workspaces with their normal Web browsers; the installation of additional software at the users' sites is not necessary. A further focus of the system is the information of the users about the activities within their workspaces, i.e., the system provides several *awareness services*.

Although the system primarily supports asynchronous modes of communication, it also provides some features for synchronous collaboration such as information about the concurrent presence of other users as well as interfaces to *synchronous communication tools* such as chat or audio/video conferencing.

3 Implementation of the BSCW System

The BSCW system is built upon a standard Web server: The Common Gateway Interface (CGI) – the standard API for Web servers – is taken to implement the BSCW kernel functionality, thereby extending a Web server into a *BSCW server*. The system is written entirely in the interpreted programming language Python (see http://www.python.org/) and the only additional software required to use the system besides a Web server is the Python interpreter.

Since Python provides good support for modularisation, the implementation of the kernel functionality and the user interface are largely separated, i.e., without modifications of the kernel code the interface can be customised to a large extent, even by people without detailed understanding of the code. The interface definition comprises a set of HTML template pages which can be edited easily. GMD provides these interface template pages in German and English, but users of the system have translated them to provide interfaces in additional languages (e.g., French, Italian, Spanish, Finnish, Russian).

The modular system design also allows extension of BSCW in a number of different ways rather easily. New operation handlers can be added to provide new functionality or act as interfaces (“wrappers’)’ to an existing application. It is also straightforward to access the persistent store of the BSCW system to store new kinds of objects without modifying the storage routines themselves. In particular, the choice of the interpreted language Python as the implementation language directly supports rapid prototyping.

An overview of the architecture of the BSCW system is given in Fig. 1. The main interface between the BSCW Server and the BSCW clients – these are normal Web browsers – is HTTP and HTML. Since HTML is not very powerful with respect to interface design, the system contains also an additional Java based interface (using XML) which has been released with version 3.3 of the system in June 1999. (This interface is described below in more detail; see also [4].)

Besides the BSCW server, i.e., a Web server extended with the BSCW functionality, the BSCW system comprises also a so-called *event server* which feeds the so-called *monitor applet* – a Java applet which can be started from a BSCW workspace – with events about presence and activities of other BSCW users (see below). This is a separate server whose functionality cannot be added to a normal Web server since HTTP and HTML are unsufficient for these particular features.

The BSCW system runs on Windows NT and various Unix dialects (including Sun Solaris and Linux). As the underlying Web server the Microsoft Internet Information Server, the Apache server and the AOL and CERN Web servers can be used.

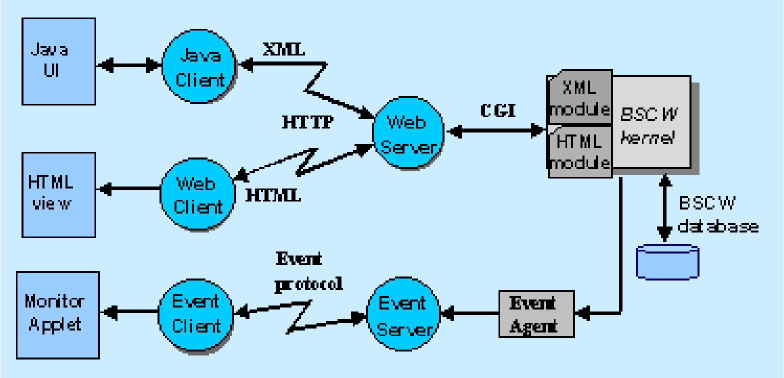


Fig. 1. Architecture of the BSCW System

4 Functionality of the BSCW System

As mentioned already above, the central metaphor of the BSCW system is the shared workspace. In general a BSCW server will manage workspaces for different groups. Users may be members of several workspaces (e.g., one workspace corresponding to each project a user is involved in). In addition, users may set up private workspaces which they do not share with others.

A shared workspace can contain different kinds of information such as documents, pictures, URL links to other Web pages or FTP sites, threaded discussions, information about other users and more. The contents of the workspaces are usually arranged in a folder hierarchy based on structuring principles agreed upon by the members of a workspace.

A cooperative system has to provide awareness information to allow users to coordinate their work. The event services of the BSCW system provides users with information on the activities of other users, with respect to the objects within a shared workspace.

Events are triggered whenever a user performs an action in a workspace such as uploading a new document, downloading ('reading') an existing document, renaming a document and so on. The system records the events and presents these events to each user in various forms, e.g., as event icons attached to the objects, via email or through messages in the monitor applet (see below).

The most common way of informing users about events is through the event icons attached to objects. Such an icon indicates that a particular event has occurred recently. Recent in this context means events which have occurred for an object since the user last carried out a *catch-up* action, an operation by which users can tell the system that they are aware of the events that have occurred so far and no longer wish to see them (i.e., their event icons) in the workspace. Events can be caught up at different levels, from individual objects to complete workspace folder hierarchies. Event histories are, of course, personal to each particular user, e.g., an event may be new to one user but old to other users.

The system distinguishes five types of events which are represented by five different event icons accordingly:

– *New events* indicate that an object has been created since the user last caught up.
– *Read events* show that an object has been downloaded or read by someone.
– *Change events* indicate that an object has been modified. This category includes several event types, such as *edited*, *renamed*, and so on.
– *Move events* show that an object has changed its location. This category includes *delete* and *undelete* events (showing the object has been moved into or out of a wastebasket) and *cut* and *drop* events (showing the object has been moved into or out of a user's personal bag.
– *Touch events* are displayed for a container such as a folder to show that something has happened to an object contained inside (either directly or lower down in the folder hierarchy).

Each event entry describes what was done, when and by whom. Although this approach for providing group awareness seems very simple at first sight, information such as "User A uploaded a new version of document X", or "User B has read document Y" is often very useful for group members in coordinating their work and gaining an overview of what has happened since they last contacted the BSCW server.

Furthermore, the system contains the following main features:

– *Authentication*: Users have to identify themselves by name and password before they have access to BSCW workspaces.
– *Version management and locking*: Documents within a workspace can be put under version control which is particular useful for joint document production, or they may be locked during an editing session to prevent other users from accessing documents temporarily.
– *Discussion forums*: Users may start a discussion on any topic they like and the system presents the threads in a style similar to the Internet newsgroups.
– *Access rights*: The system contains a sophisticated access rights model which allows, for example, that some users may have complete control over an object in a workspace whereas others have only read access or no access at all.
– *Search facilities*: Users can specify queries to find objects within BSCW workspaces based on names, content or specific properties such as document author or document modification date. Furthermore, queries may be submitted to Web search engines and the result of the query can be imported into workspaces.
– *Sorting*: As mentioned above, objects in a BSCW workspace can be ordered in a hierarchical structure according to the user requirements. Within a folder listing users may sort the objects according to several categories such as type, name, or date.
– *Document format conversion*: These facilities allow users to transform a document into their format of choice, e.g., a proprietary document format into HTML, before downloading it.

- *Annotation and rating*: Users may add notes (meta-information) to objects in a workspace and rate the quality of objects, e.g., of documents or URLs that have been created. When several users gave their rating, the system will compute a median value out of their individual ratings.
- *Upload and download of archives*: Rather than uploading documents one by one into a BSCW workspace, users may upload an archive such as a zip or tar file and extract the archive at the server which may reduce upload times significantly. Similarly, users may create an archive containing objects of a workspace and then download the archive instead of the individual objects.
- *Email integration*: Users may easily send email to other users of a BSCW server and can distribute documents in a BSCW workspace to specified recipients via email.
- *Special support for meetings*: The systems allows the creation of so-called *meeting objects* which are particularly useful for the preparation of meetings since they include features such as selection of participants, automatic invitation of participants who may accept or decline an invitation, or the distribution of meeting notifications via email.
- *Interface to synchronous communication*: Through this interface users can specify synchronous sessions and launch respective tools, e.g., audio/video conferencing software or shared whiteboard applications.
- *Anonymous access and moderation*: Anonymous access can be allowed to individual objects or complete folders, e.g., for *publishing* documents after they have been developed within a closed group. The access to public folders can be set up in such a way that users can upload documents anonymously but that they only become visible to others after they have been approved by a *moderator*.
- *Address book and calendar*: Besides the waste basket and the bag there are two further objects which are personal to each user: the address book where a user may collect the names of other (e.g., frequently contacted) users, and the calendar which contains the dates of all meeting objects related to the user.
- *Customisation*: Through user preferences the users can modify the system interface to some extent, e.g., whether or not they want to use an Javascript or ActiveX enhanced interface, and which functions they want to have available in the user interface (see also below).
- *Multi-language support*: The interface of the system can be tailored to a particular language by straight-forward extensions. Several languages (e.g., French, Italian, Spanish, Catalan) have been created by users of the system and are publicly available. Each user may select his or her preferred interface language.
- *Administration and configuration*: For administrators of a BSCW server there exists a convenient HTML interface for system administration, e.g., configuration of the server or user management. A BSCW server is highly configurable through a set of configuration files which tailor the user inter-

face of the system to particular requirements, e.g., the set of functionality which shall be accessible for the users.

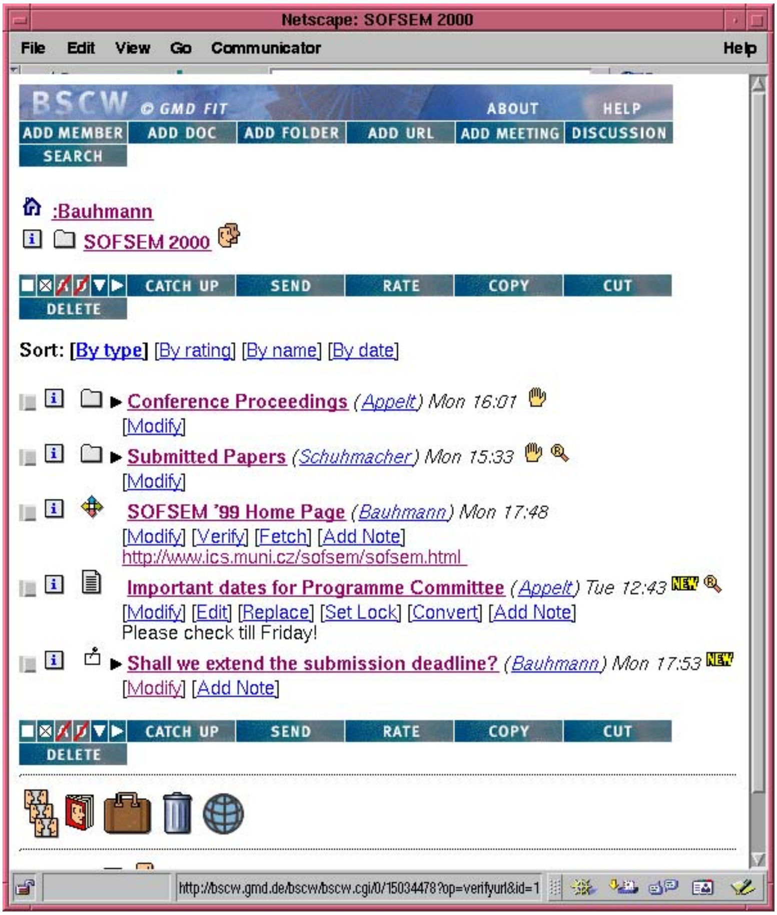
The system comprises a rich set of functions, most of which had been introduced because of user requests (see below). Many features may not be needed by all users, e.g., because they are rather specific and may, for example, only be of interest to workspace administrators. Therefore, the system supports the concept of user profiles. Users can choose between a *Beginner*, *Advanced* and *Expert* profile. In the *Beginner* profile only a subset of the functionality is visible in the interface which reduces the number of buttons and makes it thereby easier to comprehend for novice users. In the *Advanced* profile, which a user might select after he or she has become familiar with the system, the functionality is increased and more buttons appear in the interface accordingly. In the *Expert* profile the full functionality of the system is available, often only "one mouse click away", but on the expense of a rather complicated interface. Furthermore, each user may create his or her own interface by starting with one of the three predefined profiles and then adding or removing buttons to the particular requirements of the respective user. (More details are given in [1].)

In addition, there are a number of other tools contained in the BSCW system, for example, so-called uploaders, applications that transfer a file from a local file store into a particular location on a BSCW server. These tools provide more support for file uploading than is currently built into Web browsers. For example, they allow multiple file transfer or drag-and-drop uploading.

Figure 2 is an example of the user interface of the BSCW system. It shows a listing of the folder "SOFSEM 2000" for user "Bauhmann". The folder contains two sub-folders ("Conference Proceedings" and "Submitted Papers"), a link to another Web page ("SOFSEM '99 Home Page"), a text document ("Important dates for . . . ") and a discussion object ("Shall we extend . . . "). The icon in front of each object's name indicates the type of the object. Behind each object is the name of the person who created the object and the date when it was created or most recently modified.

At the top of the screen there are buttons for triggering operations such as "Add Member" to provide access to this folder to other persons, or "Add Document", "Add Folder", "Add URL", etc., to create new objects within the folder. Other actions such as "Catch up", "Send", "Rate" or "Copy" can be applied simultaneously to a group of objects which have been marked through the tick boxes in front of each object's name. Further action buttons appear in a line below each object (e.g., "Modify", "Verify", "Fetch", "Add Note", "Edit" or "Replace") since they are only applicable to one particular object.

Behind four objects ("Conference Proceedings", "Submitted Papers", "Important dates for . . . ", and "Shall we extend . . . ") there are event icons which indicate that events occurred recently, e.g., the objects "Important dates for . . . " and "Shall we extend . . . " are new for user Bauhmann, the document "Important dates for . . . " and some other document(s) within the folder "Submitted Papers" have been read and there are some further changes in the folders "Conference


Netscape: SOFSEM 2000
File Edit View Go Communicator Help
BSCW © GMD FIT
ABOUT HELP
ADD MEMBER ADD DOC ADD FOLDER ADD URL ADD MEETING DISCUSSION
SEARCH
CATCH UP SEND RATE COPY CUT
DELETE

:Bauhmann

SOFSEM 2000

Sort: [By type] [By rating] [By name] [By date]

▶ Conference Proceedings (Appelt) Mon 16:01
[Modify]

▶ Submitted Papers (Schuhmacher) Mon 15:33
[Modify]

SOFSEM '99 Home Page (Bauhmann) Mon 17:48
[Modify] [Verify] [Fetch] [Add Note]
http://www.ics.muni.cz/sofsem/sofsem.html

Important dates for Programme Committee (Appelt) Tue 12:43 NEW
[Modify] [Edit] [Replace] [Set Lock] [Convert] [Add Note]
Please check till Friday!

▶ Shall we extend the submission deadline? (Bauhmann) Mon 17:53 NEW
[Modify] [Add Note]

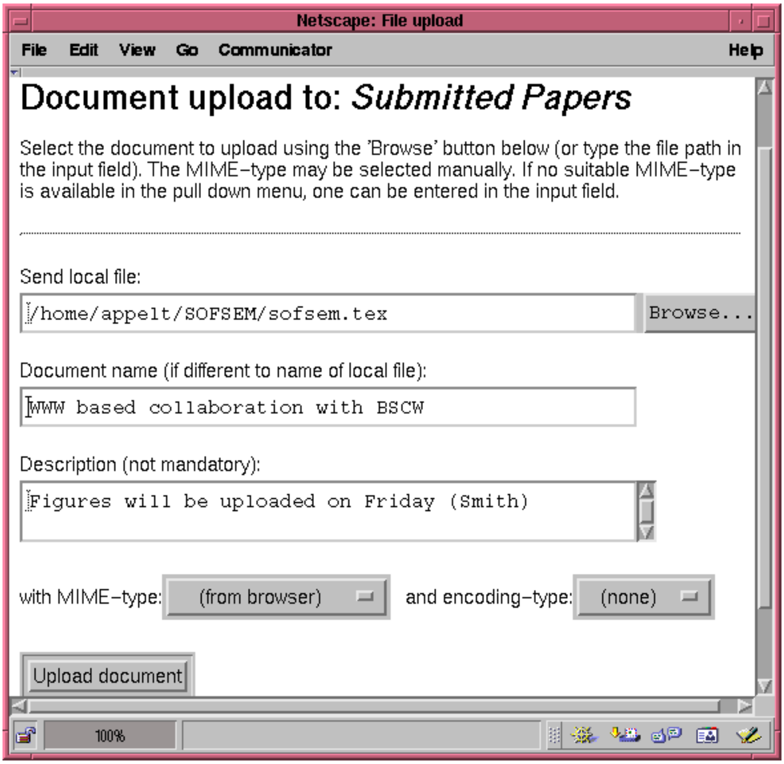
http://bscw.gmd.de/bscw/bscw.cgi/0/15034478?op=verifyurl&id=1

Fig. 2. HTML user interface to a BSCW shared workspace

Proceedings" and "Submitted Papers". Clicking on these event icons would give more details about the event, e.g., which user(s) caused the respective events.

Figure 3 gives another example of the user interface. Here a form is shown which the user has to fill in when he or she wants to upload a document into a BSCW workspace. The user has to select the file from the local file system (`/home/appelt/SOFSEM/sofsem.tex`), may specify a different name for the document on the BSCW server ("WWW based collaboration with BSCW"), and may also add some additional information to the document ("Figures will be ..."). If the Web browser is not able to determine the MIME type of the document correctly, the user may set the MIME type explicitly and specify an encoding, if applicable.

The HTML based listing of the content of folders may look a bit unusual to novice users of the system but is mainly caused by the limitations of HTTP and



Document upload to: *Submitted Papers*

Select the document to upload using the 'Browse' button below (or type the file path in the input field). The MIME–type may be selected manually. If no suitable MIME–type is available in the pull down menu, one can be entered in the input field.

Send local file: /home/appelt/SOFSEM/sofsem.tex Browse...

Document name (if different to name of local file): WWW based collaboration with BSCW

Description (not mandatory): Figures will be uploaded on Friday (Smith)

with MIME–type: (from browser) and encoding–type: (none)

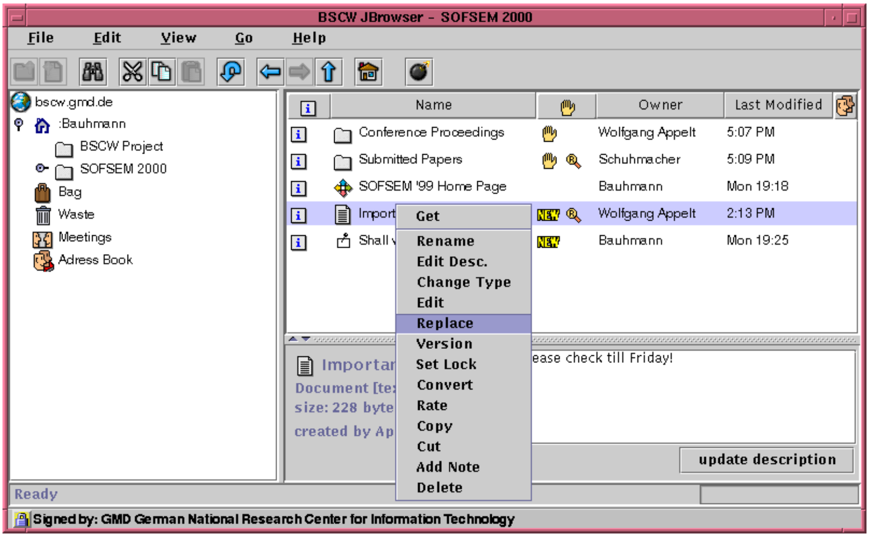
Upload document

Fig. 3. HTML form for uploading a document

HTML. The majority of users is today familiar with modern desktop graphical users interfaces such as Microsoft Windows or Macintosh OS. These interfaces contain also features such as *drag-and-drop* and pop-up menus that appear when clicking, e.g., on the mouse buttons. Unfortunately, the Web browser interfaces are not really graphical but primarily text oriented interfaces because the origin of HTML is mark-up of text.

To provide a more convenient and more familiar interface to the BSCW system, we therefore developed a Java applet which provides an additional BSCW interface. A user may launch this applet from his or her BSCW start page and access the full functionality of the system starting from this interface. At present, the functionality of the applet is primarily focussed on browsing through folder hierarchies in BSCW workspaces and the traditional HTML interface is still deployed for a larger number of operations (e.g., for filling the respective data into the form sheet used for uploading a document or creating a new link object), i.e., only a subset of the BSCW functionality is currently fully integrated in the Java applet.

Figure 4 gives an example of the Java based interface showing essentially the same content as Figure 2. This interface looks much less cluttered as the HTML



	Name		Owner	Last Modified
i	Conference Proceedings		Wolfgang Appelt	5:07 PM
i	Submitted Papers		Schuhmacher	5:09 PM
i	SOFSEM '99 Home Page		Bauhmann	Mon 19:18
i	Import	NEW	Wolfgang Appelt	2:13 PM
i	Shall v	NEW	Bauhmann	Mon 19:25

Importar
Document [te:
size: 228 byte
created by Ap

ease check till Friday!

Signed by: GMD German National Research Center for Information Technology

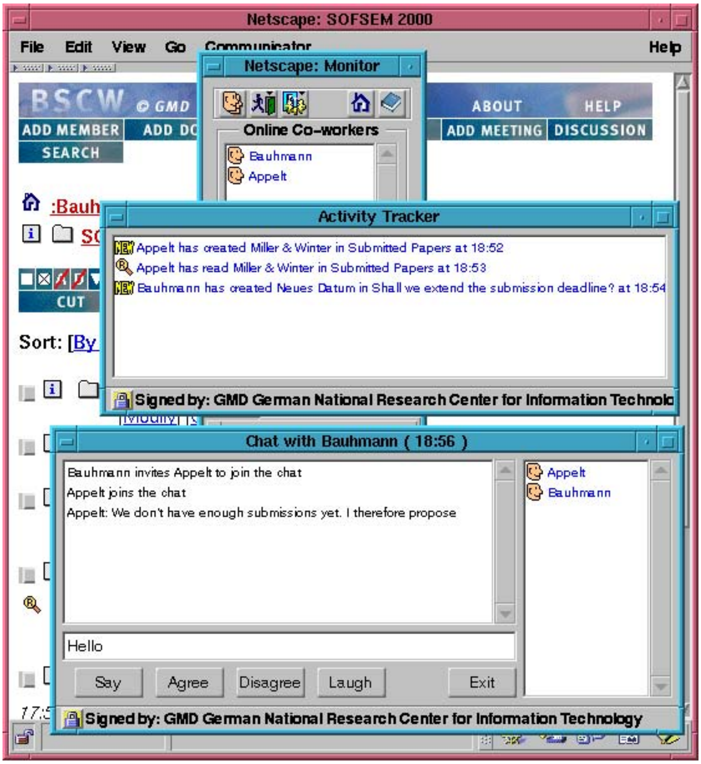
Fig. 4. Java browser interface to a BSCW shared workspace

interface because most of the action icons and buttons have been moved to pop-up menus which appear when clicking on mouse buttons. As Figure 4 shows, clicking the right mouse button when pointing to a document displays a menu with the operations applicable to documents ("Get", "Rename", . . . , "Delete"). The list of operations in the menus are filtered against the access rights of the user, i.e., only those operations are shown which are allowed for the respective user.

Although primarily focussed on asynchronous modes of cooperation, BSCW provides also some features to enable synchronous communication, e.g., the *monitor applet* mentioned above which is connected to the *monitor server* (see Figure 1). BSCW server and monitor server communicate which each other: the BSCW server informs the monitor server about the events which the monitor server then distributes to the respective Java applets (see [5] for details).

The monitor applet includes the following features: When a user starts this applet, it will show other users who have also launched the applet and who are included in the user's personal address book. (It is assumed that users belonging to the same group – i.e., those who cooperate in some way – include each other into their address books.) The applet can also indicate the activities of users visible in the applet. Furthermore, it can be used to start a chat session or send a message to other users.

Figure 5 shows an example of the monitor applet with three windows where one window shows that the users Bauhmann and Appelt are currently working with the BSCW server. The second windows shows their activities and the third window has been launched by Bauhmann for a chat session with Appelt.



Appelt has created Miller & Winter in Submitted Papers at 18:52
Appelt has read Miller & Winter in Submitted Papers at 18:53
Bauhmann has created Neues Datum in Shall we extend the submission deadline? at 18:54

Bauhmann invites Appelt to join the chat
Appelt joins the chat
Appelt: We don't have enough submissions yet. I therefore propose

Hello

Fig. 5. Monitor Applet

5 Usage of the BSCW System

At a very early stage, we decided to test our ideas and developments in a real world setting: When the first version of the system was ready in October 1995, we made it publicly available on one of GMD's Web servers (http://bscw.gmd.de/) and invited all interested people to use the system for group cooperation. Furthermore, we also made the code of the system available for download so interested parties could install their own BSCW server. (Licences for the System are now available from OrbiTeam Software GmbH, a spin-off company founded in 1998. Schools and universities can usually receive a royalty free license for educational purposes.)

In fact, we attracted several hundred users within a few weeks and soon received some quite considerable number of emails with respect to feedback on problems that users had with the system or improvements and extensions they wanted. Therefore, we decided that the future development of our BSCW system should be informed to a large extend by this feedback since this seemed a very promising approach to get a high acceptance from our user community. Since 1996, some additional funding for the development has been received from the European Commission's *Telematics Applications Programme* through the CoopWWW (1996/1997) and CESAR (1998/1999) projects.

Version 2.0 of the system was released in August 1996, version 3.0 in June 1997, and the most recent version 3.3 in June 1999. At present (August 1999) there are over 20,000 registered users at GMD's public BSCW server. On an average day, there are about 30,000 requests from users resulting in a data transfer of about 300 Megabyte.

The BSCW server software has been downloaded over several thousand times and we know of several hundred operating BSCW servers all over the world – many of them at universities – so we estimate that there exist several ten thousand BSCW users world wide.

A systematic evaluation of the usage of the BSCW system has not been carried out and is probably impossible considering the large number of users. We know, however, a number of different application areas where the system is used, e.g., project management in large projects with members from different organisations, conference organisation including handling of the paper review process, teleteaching applications including also some cases where teachers and students were located in different countries, and electronic support activities between large telecommunication companies with their customers. Two examples of BSCW usage in a university environment are described in [6] and [2].

6 Related Systems

When the first version of BSCW was released in 1995, it was the first fully Web based groupware system. At that time, other (commercial) groupware systems were based on private protocols and formats, requiring major software installation and maintenance. In 1996, BSCW won the European Software Innovation Award (ESIP '96) for its new approach in groupware developments.

Within the last few years, however, a number of other groupware systems have emerged which are based on Internet and Web technology. This includes systems which have been developed from scratch such as Hyperwave [7] or Livelink [8], but also systems which have replaced – more or less thoroughly – their private protocol and format by open standards such as the most recent versions of Lotus Domino [9].

We believe that BSCW is still one of the leading systems for collaboration support. Its strength is surely based on the large feedback from its user community which contributed much to the current status of the system.

7 Conclusions

The BSCW shared workspace system is a Web-based CSCW tool offering a wide range of features to support collaboration. In particular, the system is considered a very useful tool for cooperation in locally dispersed, cross-organisational groups using different system platforms.

References

1. Appelt, W., Hinrichs, E., Woetzel, G.: Effectiveness and Efficiency: The Need for Tailorable User Interfaces on the Web; Proceedings of the 7th International WWW Conference, Brisbane, 1998. 72
2. Appelt, W., Mambrey, P.: Experiences with the BSCW Shared Workspace System as the Backbone of a Virtual Learning Environment for Students. Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications ED-MEDIA 99, Seattle, 1999. 77
3. Gorton, I., Hawryszkiewycz, I., Fung, L.: Enabling Software Shift Work with Groupware: A Case Study; Proceedings of the 27th Hawaii International Conference on System Science, IEEE Computer Society Press, 1996. 67
4. Koch, Th.: XML in practice: the groupware case. Proceedings of IEEE WET ICE Workshop 1999, Stanford University. 68
5. Trevor, J., Koch, Th., Woetzel, G.: MetaWeb: Bringing synchronous groupware to the World Wide Web. Proceedings of the European Conference on Computer Supported Cooperative Work (ECSCW'97). Kluwer Academic Publishers, 1997. 75
6. Vliem, M. E., Using the Internet in university education – The application of BSCW within student projects. Report of the Ergonomics Group, University of Twente, Enschede, 1997. 77
7. http://www.hyperwave.com 77
8. http://www.opentext.com/livelink 77
9. http://www.lotus.com 77

Middleware and Quality of Service

Christian Bac, Guy Bernard, Didier Le Tien, and Olivier Villin

Institut National des Telecommunications
9 rue Charles Fourier 91011 Evry Cedex, FRANCE
{chris,bernard,letien,villin}@etna.int-evry.fr

Abstract. This presentation is a tutorial and a state of the art in Quality of Service and support for Quality of Service in middleware. It is organized as follow:

- First, we briefly introduce the problematic of Quality of Service and middleware.
- The second part, presents concepts handled in Quality of Service to support distributed applications.
- The third part, shortly presents middleware concepts and how Quality of Service is added in existing middleware implementation. It explains which concepts are common to these implementations.
- Finally we summerize what is already done in term of QoS Support in middleware, what is expected to come, and what are the unresolved problems that need more work to overcome.

1 Introduction

This presentation is a tutorial and a state of the art in Quality of Service (QoS) and support for Quality of Service in middleware. There are four parts in the speech:

- In the first part, we briefly introduce the problematic of Quality of Service and middleware. First we show what are the basic needs of QoS Systems and why a middleware is not well structured to respond to these needs. Then we explain why we want to implement QoS demanding applications in a distributed Object Oriented System.
- In the second part, we present concepts handled in Quality of Service to support distributed applications.
- In the third part, we shortly present middleware concepts and how Quality of Service is added in existing middleware implementation.
- Finally we conclude summarizing what is already done in term of QoS Support in middleware, what is expected to come, and what are the unresolved problems that need more work to overcome.

1.1 Some Key Concepts

Let's jump into the problem first and try to analyze it in deep afterward. We try and set up a distributed system based on middleware that is able to support quality of service constraints. What does this mean:

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 79–94, 1999.

1. Middleware: is a software architecture to support distributed applications. The main characteristics of middleware are that:
   - middleware is designed to ease the building of distributed application, specially for client-server programming style.
   - an application in a middleware is defined in terms of functionalities;
   - basic interactions between a client and a server rely on remote invocation mechanism. Middleware comes with a Remote Procedure Call package (RPC), and its capacities depends on this package. This means that usual interaction in middleware is synchronous.
   - middleware provides transparency, this means that a client ignores the server localization when it uses it. In the same idea, middleware allows architecture independence between clients and servers;
   - each middleware is associated with a programming style that is more or less object oriented. For example, CORBA and JAVA/RMI are object oriented, DCE is not.
2. QoS: is the evaluation of non functional requirements. These requirements are associated to the relation between the user and the computer. QoS is linked to applications that carry complex informations that may be related to images or sounds, these kind of applications are usually called multimedia applications. This means that these requirements are subjective for many of them and also that they are tighten to the way human perception works. For example, the sound is much strict on the transmission delay than the image. These non functional requirements are described with parameters:
   - that can be temporal constraints for example the elapsed time between the moment when you hit a button and the moment when the corresponding action is undertaken by the system.
   - that may be some supportable error rate for example how many words can you miss in a sentence that does not prevent you of understanding the meaning of this sentence;
   - in the ISO terminologies, QoS is "The global performance effect on service that determines the degree of a user, that uses this service, satisfaction".

1.2 Why Mix Middleware and QoS?

To summarize:

- QoS means resources management that includes resources localization, resource usage negotiation and control;
- Middleware allows functional execution for an application on an Object Oriented distributed platform;
- QoS and Middleware: middleware is a good support to develop distributed applications. Why wont we use it to develop distributed multimedia applications? If we do, we need a QoS aware middleware.

Now that we have shortly described the theme, lets investigate in deeper what Quality of Service means and how it is usually added in distributed systems.

2 QoS in Distributed Multimedia Systems

This section describes fundamentals in QoS.

2.1 Layered Model

As shown in figure 1, a distributed QoS system must support QoS at different layers [14]:

user: this level allows the user to specify global parameters such as good quality, standard quality or low quality. It can be associated with a graphical user interface;

application: the application must be able to quantify the QoS specified by the user and negotiate parameters with the system. It uses the user specification to calculate data size, rate and error. For example, it translate the "good image" selection in full color, 300 × 300 pixels, 30 images per seconds.

system: when the system negotiates the quality of service with the application, it translates the application parameters in values for the resources it manages. This is called the resources mapping. It can be split into two parts:
- the requirements on the communication services;
- the requirements on the operating system services on each site.

device: the system is in charge of the allocation of the resources corresponding to devices that can be:
- multimedia devices on the site;
- and devices connected to the network.

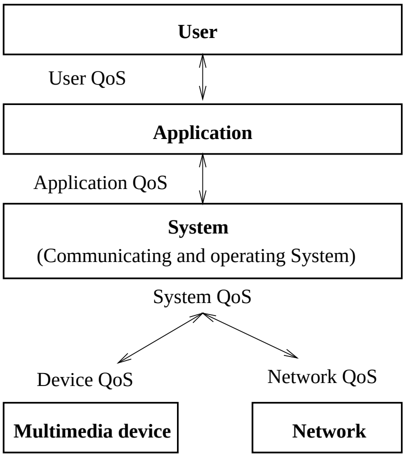


Fig. 1. Layered model

Figure 2 shows that resources management must be included at each level using a resource management protocol and negotiating the necessary resources with the lower layer. In this figure, there is also a special case for the network layer that needs a QoS aware router.

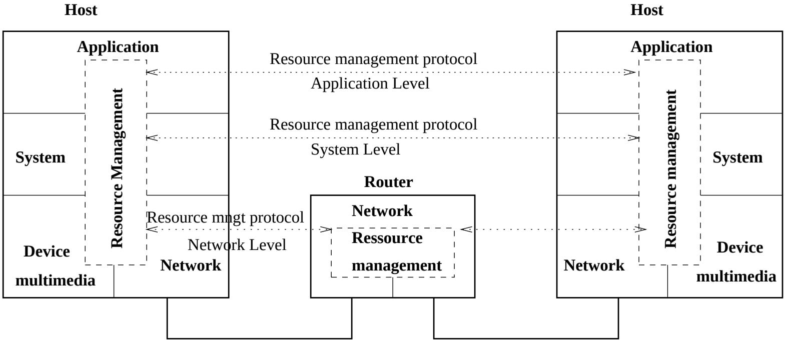


Fig. 2. Managing resources in a layered model

2.2 Main Operations

The main operations in a QoS system are:

Reservation: that allows resources allocation, and multimedia stream setting according to the application's QoS requirements;

Distribution: of resources between the application which is the usual operating system work to control the resources sharing;

Adaptation: despite the fact that resources are reserved, the system must control resources usage during execution, because the global system is shared with non-QoS applications or because components do not behave according to their commitment.

2.3 QoS Specification

To allow QoS negotiation, resources reservation and monitoring, the QoS needs must be according to the following points:

stream synchronization: describes the level of synchronization between related streams. The well known example is the lip-speech synchronization that is necessary to synchronize sounds and images in a video film.

stream performance: is directly related to resources reservation because it gives a description in terms of throughput, delay, jitter and admissible error rate.

level of guaranty: specifies the capacity of the system to ensure the QoS negotiated (end to end). It is usually split into three levels: hard guaranty that means no contract violation, soft guaranty that means average contract compliance, and no-guaranty that you can also call best-effort.

QoS policy is the description of the system's behavior in case of a contract violation at execution time.

service cost is the price to pay for the service. This is to put a brake on maximal service demand!

2.4 QoS Mechanisms

To allow QoS support the following mechanisms must be completed by the system:

QoS providing: which allows static resources management, at the stream initiation. It is split into three actions:

- **QoS mapping** that translates the user QoS requirements into the different layers;
- **admission test** that tests the translated QoS requirements toward the available system resources ;
- **reservation protocol** that reserves the resources if the admission test is successful. This is usually an end to end allocation.

QoS control: controls the data stream during the data transfer. Its main components are [3]: the **flow scheduling** that schedules the different actions according to the QoS accepted, the **flow regulation** that verifies that the data stream conforms to its specification, **flow synchronization** that controls the timing and ordering of operations and **flow control** that may allow the sender to adapt to the recipient speed (breaking the QoS reservation).

QoS management: monitors and services the resources during the data transfer to respond to system modifications. It does the **monitoring** and **servicing** between layers. In case of QoS **degradation** it does the **signaling**, and tries to manage system dysfunctions and to **adapt** to continue execution.

2.5 Communication Layers

This is where the QoS support is the most advanced, there has been early work on providing QoS at the network and transport layer. There is also ongoing work on future protocols and QoS.

2.5.1 Network Layer. The network layer must provide large bandwidth, and multicast delivery. It must allow resources reservation, and provide QoS guarantees and routing protocols to support streams.

There has been a number of reservation protocols developed and also a proposition for an IP architecture that allows packet switching according to QoS classes:

Stream Protocol Version 2 [23]: allows guaranteed service, is connexion oriented, and based on a stream model. This protocol makes resources reservation at connexion time.

Real-Time Internet Protocol [20]: specifies an unreliable datagram delivery, that is connexion oriented with performance guaranty. It uses the Real-time Channel Administration Protocol to reserve resources.

Integrated services [19]: specifies how to provide QoS on a per flow basis in the Internet. It can be mixed with RSVP to reserve resources along the data path.

Differentiated services [8]: specifies a packet stamping method that allows packet switchers to manages priority queues according to the packet stamp; this protocol avoids resources reservation.

2.5.2 Transport Layer. Transport layers have been designed to meet the needs of continuous media. The Esprit OSI 95 project proposed:

1. an Enhanced Transport Service TPX [5],
2. that is able to manage QoS parameters at connexion time like the throughput, delay, jitter and error control,
3. this transport also allows QoS semantics like *compulsory, threshold* or *maximal quality*).

The Tenet group at UCB proposed a Real-time Message TP and Continuous Media TP [12].

The University of Lancaster also developed a Multimedia Enhanced Transport Service [21]. It uses an ATM network, and provides a communication service that is connexion oriented, and guarantees that packets are ordered, but that is not reliable. This transport service does resources allocation.

2.6 QoS Architectures

To allow QoS from application to application, research group proposed architectures that support QoS at the different layers for example:

1. Integrated Multimedia Communication Architecture: from Nicolaou at the Cambridge University [15,16]. This architecture focussed on managing the multimedia data flow in an operating system.
2. Quality of Service Architecture [4] from the Lancaster University. will be reviewed as an example in the next section, it supports the QoS properties for multimedia applications.
3. Tenet UCB project developed a protocol family for wide ATM network [9].
4. and the HeiProject, in the IBM European Networking Center in Heidelberg [10,25].

2.7 Platform Example: QoS-A

QoS-A (see figure 3) is a complete example of a QoS architecture that supports distributed multimedia applications. It allows QoS specification and multimedia communication in an object oriented environment.

It exhibits the following key notions:

flows is the data path from a source (data producing) to a sink (data consuming);
service contracts is the binding between users and providers.
flow management allows QoS control and servicing.

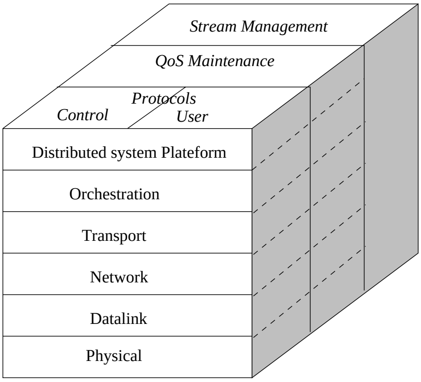


Fig. 3. QoS-A Architecture

The architecture is organized according to the following layers:

Distributed System Platform allows execution of distributed applications with services to provide multimedia communications and QoS specification.
Orchestration provides the synchronization between related multimedia data flows and jitter correction.
Transport provides configurable services and mechanisms.
Network is the base to an end to end support over an ATM network.

The platform is not only structured by layers but also by planes. The *protocol* plane is split into two sub-planes the control plane is reliable, full duplex and slow and the user plane is unreliable, one way and wide band. The *maintenance* plane does the global management. It is in charge of the monitoring and servicing in each layer. The *stream management* provides an end-to-end admission control, is in charge of the QoS mapping, and the adaptation.

QoS-A uses a Service Transport Contract that describes the commitment between the application and the system. This contract contains the following informations:

1. the QoS specification is described in terms of throughput, delay, jitter and error rate.
2. the level of agreement allows some flexibility in the commitment between the two parties. It can be deterministic (firm requirements), adaptive (statistic) and best effort.
3. the adaptation policy offers options like filtering and adaptation. It describes actions to be taken in case of contract violation.
4. the level of monitoring and servicing.
5. the reservation method that can be on demand, fast or advanced.
6. the cost specifies the service price.

2.8 Conclusion QoS

To summarize the following points are mandatory to allow QoS control in a distributed multimedia environment:

Qos Specification at the user level must be mapped in the different layers.

Service Level specifies the system commitment to the user QoS. It is often: *best effort*, *adaptive* or *guaranteed*.

QoS management must take place at different moment:
- *static*: at the admission control, it does the resources reservation and QoS mapping.
- *dynamic*: during the data transmission, it does the monitoring, control, adaptation and servicing.

Connection is the abstraction used to model a media stream. It describes a data path, and the associated resources reservation.

3 Middleware

In this section we shortly describe what a middleware is specially what is a CORBA environment.

3.1 Characteristics

A definition for middleware is:

Distributed Environment for applications re-usability, portability and inter-operability.

The common points of middleware are:

- the use of an Interface Definition Language that allows the description of client and server interactions. The IDL is aimed at been language and implementation independent.
- every middleware is based on a Remote Procedure Call package that it uses for the interactions between clients and servers. The RPC package is also associated with a naming service that allows the communication to be location independent (i.e. client and server ignore the location of their relative).

– some middleware allow execution control, for example DCE comes with its threads package.
– middleware add specifics functions that relies on the RPC and naming services, for example CORBA defines trading and persistancy services. DCE defines a file system service.

The rest of this presentation is mainly based on CORBA due to the fact that the most active middleware and QoS community acts in CORBA environments.

3.2 OMA Architecture

CORBA is based on a general architecture called the Object Management Architecture [24]. In the OMA, an object is an entity that can be accessed through an interface. This architecture, see figure 4, is based on:

the Object Request Broker (ORB) that enables communication between clients and objects;
the Object Services offers interfaces that are of general purpose usage to create services, for example the trading service and the naming service are object services;
the Common Facilities this facilities are also of general purpose usage but are more oriented toward end-user applications;
the Domain Interfaces these interfaces completes the Object Services and Common Facilities but for a specific domain for example Product Data Management or Telecommunications;
the Application Interfaces are specifically developed for an application.

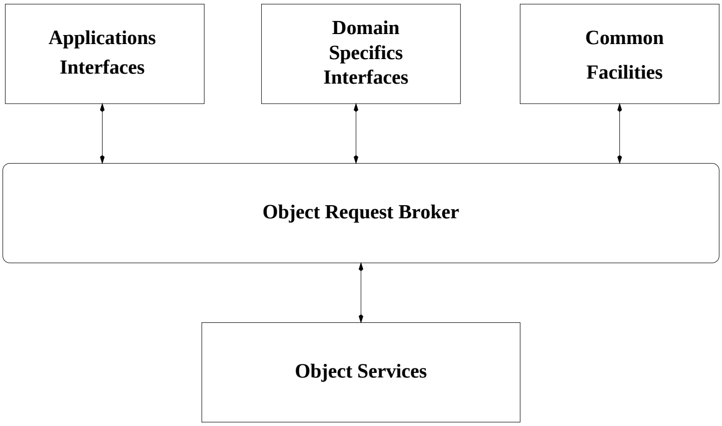


Fig. 4. OMA Architecture

3.3 CORBA Architecture

The Common Object Request Broker Architecture (CORBA) [18] details the interfaces and characteristics of the ORB in the OMA. The main features shown in figure 5 of CORBA 2 are:

- ORB Core
- OMG IDL
- Interface Repository
- Language Mappings
- Stubs and Skeletons
- Dynamic Invocation and Dispatch
- Object Adapters
- Inter-ORB Protocols.

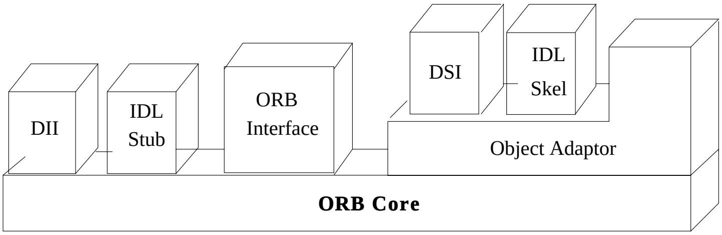


Fig. 5. ORB Architecture

The component that is of great interest in middleware and QoS is the ORB core, that we describe in the next section.

3.4 ORB Functionalities

The ORB delivers requests to objects and return responses to the client making the request. To achieve transparency, an ORB hides the following:

- Object location: The client does not know where the target object resides.
- Object implementation: The client does not know how the target object is implemented.
- Object execution state: The client does not need to know whether the target object is already started or not.
- Object communication mechanism: The client does not know what communication mechanism the ORB uses to deliver the request and return the reply.

As already stated, the fact that the ORB is hiding so many mechanism is a brake to the support of QoS in the CORBA architecture, we will now look at some architectures that try to support QoS in CORBA.

4 QoS CORBA Architectures

The following research group are working on providing QoS in a middleware environment based on CORBA.

- the Lancaster University that works on middleware and reflexivity [2];
- the BBN Systems and Technologies laboratories that tries to provide a QoS middleware by federating research projects in the USA [26];
- the Rhode Island University, that works on Real Time [27];
- the ReTINA consortium that develops a distributed processing environment for telecommunications [6];
- the Center for Distributed Object Computing G. Washington University in St. Louis Missouri that we will present as an example of middleware providing QoS support.

4.1 Example: TAO

Figure 6 shows the TAO [22] architecture as an example of what must be done to CORBA to support real-time invocations and QoS support.

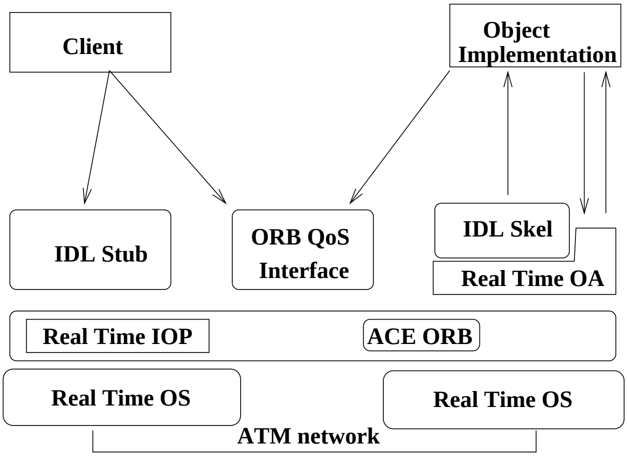


Fig. 6. TAO Architecture

This implementation uses an ATM network and real time operating systems. The architecture modifications includes:

1. speedup in the stub and skeleton;
2. usage of an ORB based on specific communication mechanisms called ACE;
3. Taylor ed specific ORB interface to support QoS;
4. special real-time inter ORB protocol to communicate with the other sites;
5. real-time object adaptor that is priority aware.

The following subsections gives some comparisons about different research projects that try to mix QoS and CORBA.

4.2 Comparison Tables

Table 1 mentions what is the main goal of the project and states whether it seemed necessary to modify the IDL to express the QoS constraints in this project. We can deduce from this table that even if the OMG states that it wont allow IDL modification there is a need for this modification at least to express some kind of real-time constraints.

Table 1. IDL modification

Project	Aims	extension IDL
RHODES	Database et CORBA Real Time	yes
TAO	real time	yes
ReTINA	interactive multimedia services	yes
QuO	objects QoS	yes
Lancaster	multimedia communication	yes

Table 2. ORB extensions in Corba RT projects

Project	ORB Modif.	Services
RHODES	no	priority and globals times, concurrency control, events
TAO	RT RIOP	RT scheduler, events
ReTINA	telecom ORB	info, telecom
QuO	not finished	QoS delegation
Lancaster	yes	interface type verification events, QoS manager, QoS mapper

Table 2 shows whether it was necessary to modify the ORB to support real-time and QoS constraints. It also show what new services have been added. This table shows that it seems obvious that modifications in the ORB core are necessary to cover a wide range of QoS support. Some QoS support may be achieved using only new services. The first service for QoS and CORBA is an event service that is present in nearly all the implementations.

Table 3 shows if the architecture allows some kind of resources specification and what kind of operating system and network they use. Conclusions from this table is that we need to specify the resources that an application needs, we are to use a real-time operating system and a network that allows resources reservations (ATM for now).

Table 3. Point of vue system and network

Architecture	**Resources Specification**	**Operating System**	**Network type**
RHODES	no	RT	–
TAO	yes	RT	ATM
ReTINA	yes	RT	multi-protocols
QuO	yes	RT	–
Lancaster	yes	Chorus	ATM

Table 4 exhibits some noteworthy notions in the different architectures.

Table 4. Noteworthy Notions

Architecture	**Notions**
RHODES	Timed Dynamic Method Invocation, distributed RT scheduling
TAO	GIGABIT sub-system, quick marshaling
ReTINA	binding mechanism
QuO	QDL, delegate object, QoS region
Lancaster	EDL, Query Language, event, binding object

4.3 Bases for a QoS CORBA

This different architectures teach us on what are the fundamental bases to develop a QoS CORBA architecture:

- The first modification must be at the language level, we must include QoS control at the IDL level, either by modifying the OMG IDL or by adding some kind of QoS Description Language that is used to specify the application and the object server interface.
- The second modification is at the ORB level so that it becomes a QoS aware middleware.
- The third modification seems to adapt the middleware to fast networks so that the middleware is able to use the potential bandwidth. This means usually that we must find some light mechanism like Lightweight RPC to speed up the marshaling and un-marshaling, and avoid data copies through the different layers.
- A QoS middleware needs be able to schedule the activities with some kind of real-time algorithm so it must at least execute on a real-time operating system.

- A QoS middleware is only manageable over a Quality of service aware network and nowadays this means use ATM network.

Lets add some design principles to this bases.

- The QoS part of the architecture must be Object Oriented to favor portability, and give an uniform access to applications.
- The QoS negotiation must take place through the ORB and the ORB must be QoS aware so that it must reserve necessary resources.

5 Conclusion on QoS and Middleware

To summarize: middleware separates applications from resources and to support QoS one needs relations between applications and resources. For an application deployed on a middleware access to system objects hard, and the layered vision is functional so it is hard to express QoS needs.

The current practice exhibits two different types of solutions:

1. research teams that work in the spirit of the Open Distributed Processing [17]:
   (a) ODP offers meta solution to the problem for example the ODP binding objects, streams control. Try an map these solutions to your middleware.
   (b) Lancaster university [2]: proposes to introduce planes and reactivity in the ORB. Extends a micro-kernel OS to support and ORB that implements ODP bindings for multimedia streams;
   (c) DIMMA [6] uses the ANSA [1] ORB and adds an ODP library and streams support;
   (d) Jonathan [7]: is an ORB written in JAVA that includes reactive programming.
2. pure CORBA solutions are developed by teams that where not formally involved in ODP:
   (a) OMG RFP 97-05-04 proposes to manage streams through an ORB but to use a separate connection [13] to create the multimedia data path. The idea is that one must not stress the ORB that is not designed to carry a data stream.
   (b) QuO [26] proposes some adaptation mechanisms and a quality level management. The ORB implementation is not yet finished.
   (c) TAO [22] develop a fast ORB with tasks scheduling related to messages priorities. To allow multimedia support they added off line scheduling (i.e. admission test), and on line scheduling (i.e. execution control).

5.1 Future Work

For now the following points are either not consensual or not enough worked:

1. there still has no norm complete enough for describing every type of QoS,
   (a) the network level is the most advanced there are well known QoS [11] parameters and support in existing protocols (ATM, RTP);
   (b) the debate on whether to create the virtual channel to support streams in or out of middleware is not finished;
   (c) at the ORB level there is no consensus on how an application must express its QoS needs;
2. the interactions between the middleware and the system are not enough explored; QoS middleware maps the QoS constraints to a fixed priority they give to some kind of threads (usually POSIX compliant). The most advanced project on this is the Nemesis kernel;
3. on other middleware: there is no publication.

References

1. ANSA Architecture Report: The ANSA Computational Model AR.001.01. ANSA, Cambridge (UK), February 1993. 92
2. G. S. Blair et al. An Architecture for Next Generation Middleware. In *MIDDLEWARE 98*, http://www.comp.lancs.ac.uk/computing/middleware98/, September 1998. Springer Verlag ISBN 1-85233-088-0. 89, 92
3. A. Campbell, C. Aurrecoecha, and L. Hauw. A Review of QoS Architectures. In *Proc. 4^{th} Int. IFIP Workshop on Quality of Service, IWQoS96*, Paris (France), March 1996. 83
4. A. Campbell, G. Coulson, and D. Hutchison. A Quality of Service Architecture. *ACM SIGCOMM Computer Communication Review*, April 1994. 84
5. A. Danthine, Y. Baguette, G. Leduc, and L. Leonard. The OSI 95 Connection-Mode Transport Service – Enhanced QoS. In *Proc. 4^{th} IFIP Conference on High Performance Networking*, Liège (Belgium), December 1992. 84
6. D. Donaldson, F. Faupel, R. Hayton, A. Herbert, N. Howarth, A. Kramer, I. MacMillan, D. Otway, and S. Waterhouse. DIMMA – A Multi-Media ORB. In *Proc. of Middleware'98*, The Lake District, September 15–18, 1998. 89, 92
7. B. Dumant, F. Horn, F. Tran, and J.-B. Stefani. Jonathan: an Open Distributed Processing Environment in Java. In *Proc. of Middleware'98*, The Lake District, September 15–18, 1998. 92
8. S. Blake et al. An Architecture for Differentiated Services. Request for comments no rfc-2475, Internet, December 1998. 84
9. D. Ferrari. The Tenet Experience and the Design of Protocols for Integrated Services Internetworks. *Multimedia Systems Journal*, November 1995. 84
10. D. B. Hehmann, R. G. Herrtwich, W. Schulz, T. Schuett, and R. Steinmetz. Implementing HeiTS: Architecture and Implementation Strategy of the Heidelberg High Speed Transport System. In *Proc. 2^{nd} International Workshop on Network and Operating System Support for Digital Audio and Video*, IBM ENC, Heidelberg (Germany), 1991. 84
11. QoS – Basic Framework – CD Text (ISO/IEC JTC1/SC21 N9309, 1995. 93

12. E. W. Kightly and H. Zhang. Connection Admission Control for RED-VBR. In *Proc. 4th Int. IFIP Workshop on Quality of Service, IWQoS96*, Paris (France), March 1996. 84
13. S. Mungee, N. Surendran, and D. C. Schmidt. The Design and Performance of a CORBA Audio/Video Streaming Service. Technical Report WUC-98-15, Washington University, 1998. 92
14. K. Nahrstedt. *Quality of service in networked Multimedia Systems*, chapter 10. CRC press, 1999. 81
15. C. Nicolaou. An Architecture for Real-Time Multimedia Communication Systems. *IEEE Journal on Selected Areas in Communications*, 8(3), April 1990. 84
16. C. A. Nicolaou. *A Distributed Architecture for Multimedia Communication Systems*. PhD thesis, University of Cambridge (UK), May 1991. 84
17. Basic Reference Model of Open Distributed Processing, November 1992. 92
18. Common Object Request Broker: Architecture and Specification Revision 2.1. OMG Document 97-08, 1997. 88
19. D. Clark R. Braden and S. Shenker. Integrated Services in the Internet Architecture: an Overview. Request for comments no rfc-1633, Internet, June 1994. 84
20. RTP: A Transport Protocol for Real-Time Applications, 1996. 84
21. A. Smith and A. Grace. *A QoS Configuration System for Distributed Applications*, chapter 6. Chapman & Hall, London, 1997. 84
22. A. Schmidt, D. C. Gokhale, T.R H. Harrison, and G. Parulkar. A High-performance Endsystem Architecture for Real-time CORBA. *IEEE Communications Magazine*, 14(2), February 1997. 89, 92
23. C. Topolcic. Experimental Internet Stream Protocol, Version 2 (ST-II). Request for comments no rfc-1190, Internet, October 1990. 84
24. S. Vinoski. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. *IEEE Communications Magazine*, 14(2), February 1997. 87
25. C. Volg, L. Wolf, R. Herrtwich, and H. Wittig. HeiRAT – Quality of Service Management for Distributed Multimedia Systems. *Multimedia Systems Journal*, November 1995. 84
26. R. Vanegas, J. Zinky, J. Loyall, D. Karr, R. Schantz, and D. Bakken. QuO's Runtime Support for Quality of Service in Distributed Objects. In *Proc. of Middleware'98*, The Lake District, September 15–18, 1998. 89, 92
27. V. F. Wolfe, L. C. DiPippo, R. Ginis, M. Squadrito, S. Wohlever, I. Zykh, and R. Johnston. Real-Time CORBA. University of Rhode Island Department of Computer Science and Statistics Technical Reports TR97-256, University of Rhode Island, June 1997. 89

Dynamic Reconfiguration of CORBA-Based Applications

Noemi Rodriguez and Roberto Ierusalimschy

Departamento de Informática, PUC-Rio
22453-900, Rio de Janeiro, Brazil
noemi,roberto@inf.puc-rio.br

Abstract. Most current support for dynamic reconfiguration assumes that component interfaces specify input and output channels. Component models such as CORBA, however, support a client-server architecture, where component interfaces describe only the offered services. This work discusses the use of an interpreted language as a tool for dynamic configuration of distributed applications using CORBA components. We describe LuaOrb, a system based on the CORBA Dynamic Invocation Interface (DII) and the Dynamic Skeleton Interface (DSI), which provides Lua programs with easy access to CORBA servers and allows these servers to be dynamically modified. Using LuaOrb, the Lua console itself becomes a tool for reconfiguration. LuaOrb uses a structural sub-typing model, so that only correctly typed connections are accepted. We also discuss possible forms for prescribing a reconfiguration, and their relation to LuaOrb.

1 Introduction

Component-based programming has been receiving a lot of attention and is frequently considered the successor of object oriented programming. A fundamental point in the concept of component is the separation between interface and implementation. To use a component, it is necessary only to know its interface — implementations need not and, in general, should not be public. This allows client programs to remain valid even when an implementation of an invoked component is substituted for a new version.

CORBA is the most widely accepted component model for open systems, with several commercial and academic implementations [20]. However, not much attention have been given to the problem of managing change in CORBA based applications. The need for dynamically evolving applications using CORBA is becoming increasingly apparent, specially in areas such as network management and other real time control systems. Fault tolerance and the dynamic integration of newly available services are important goals in these areas, as is the avoidance of service interruption.

In CORBA, interfaces are specified using an *IDL* (Interface Definition Language). Specifications written in IDL are typically compiled into stubs (in the client side) and skeleton programs (in the server side). Modifications in the

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 95–111, 1999.

structure of the application (client-side) or in the implementation of components (server-side) require re-compilation and service interruption.

In this work, we discuss how an interpreted language can be used to add flexibility to CORBA-based programming. We present a set of tools which use alternative, dynamic mechanisms for client and server implementation and discuss how these tools can be used to support dynamically evolving applications. Tools and examples are based on the extension language Lua and its binding to CORBA, called LuaOrb.

In the next section, we define interpreted languages, and discuss their advantages and disadvantages. We describe Lua and LuaOrb in Section 3, and in Section 4 show how they can be used to solve some classic configuration problems. In the last section we draw some conclusions.

2 Using an Interpreted Language

We will adopt the following definition: a language will be said to be *interpreted* if it offers some mechanism for execution of chunks of code created dynamically; in other words, if the interpreter is directly accessible to the language. According to this definition, languages such as Lisp, Lua, Mumps and Tcl are interpreted, while Pascal, C, C++ and Java are not.

Programs are frequently developed in one environment, and later installed and configured in a different target environment. Much of this configuration activity relates to setting program variables (such as IP addresses, local directories, server names, etc.) to appropriate values. Such configuration may be done through the use of environment variables or simple text files such as the X system "resource" files. However, as programs become more complex, configuration possibilities increase, specially in interface-related issues. Many programs allow menus to be created or modified, and even new operations (macros) to be defined. To support such flexibility, configuration must many times be controlled by a full fledged programming language. This need fueled the development of *extension* languages such as Tcl [16] and Lua [8].

With the incorporation of an interpreter to the run-time environment, program configuration files can contain much more than a list of data and options. A configuration file can contain initialization routines which use all the expressive power available in a programming language (conditionals, loops, abstractions, etc).

In the preceding paragraphs we have been using the term "configuration" in the sense of tailoring an application to specific needs and environments. However, this task is not so different in its goals from the configuration of distributed applications. Rather, these are different points in a continuum of complexity levels. Therefore, it makes sense to explore the use of a language such as Lua, that has been proving itself very useful for the extension and configuration of sequential applications, as a tool for the configuration of distributed applications.

In the context of configuration of applications, the fact that interpreted languages allow changes to take place with no need for re-compilation becomes specially interesting, as it allows distributed applications to be dynamically modified with no need for service interruption. The interactivity offered by an interpreted language also fits in well with configuration requirements for testing and prototyping: the programmer can use a console to test components and partial configurations directly, avoiding the need for test stubs.

Two weaknesses generally associated to interpreted languages are efficiency and robustness. In fact, interpreted languages are normally much slower than their compiled counterparts (a factor of 10 is not uncommon). However, in component-based applications, components may be coded in compiled, efficient languages such as C, with the interpreted language acting only as a a flexible connection element. In this role, the performance penalties interpretation impose may be overruled by time spent on communication or input/output calls. Similarly, robustness in a language must be evaluated in the context of its use. Static verification is certainly an important ally in the development of large software projects. Yet conventional programming languages pay a price for static typing, namely, the loss of polymorphism and flexibility. Besides, if on one hand static typing is not available, on the other hand interpreted languages usually rely on sturdy run-time error-checking mechanisms (for uninitialized variables, dangling references, etc) which may be very useful for program debugging. In the case of CORBA based applications, it is important to emphasize that each CORBA component will typically have been developed with a conventional, statically typed language. The use of an interpreted language as a *glue* between these components may result in a run-time error if an attempt is made to invoke a non-existing method or to call a method using incorrect parameters. As explained in the next section, such situation will generate *fallbacks* in Lua, which are in some ways similar to exceptions, and may be appropriately handled for program flexibility.

3 LuaOrb

LuaOrb is a binding between CORBA and the language Lua, an interpreted language developed at PUC-Rio [8,5]. Lua is an *extension* language, implemented as a library. With its API, it is very easy to call Lua functions from C code, as it is very easy to register C functions to be called from Lua code.

The CORBA standard [15] provides communication facilities to applications in a distributed environment. All communication between CORBA objects is mediated by the *Object Request Broker* (ORB) [15,21]. A client can interact with the broker through stubs or through its *Dynamic Invocation Interface.*

OMG IDL is a technology-independent syntax for describing object interfaces. Typically, specifications written in IDL are compiled into client stubs (called simply stubs) and server stubs (called skeleton programs). The client program is directly linked to the stub. The server program must implement the methods declared in the skeleton.

This approach, used in most current language bindings, such as C++, Java, and Smalltalk, requires clients to be recompiled each time a change in the server's interface takes place or each time a new type of object is to be used as a server. Servers must be recompiled after any modification to their interface or implementation.

The CORBA architecture offers two mechanisms which allow programs to circumvent this need for re-compilation. On the client side, the Dynamic Invocation Interface (DII) is a generic facility for invoking any operation with a run-time–defined parameter list. On the server side, the Dynamic Skeleton Interface (DSI) is an interface for writing object implementations that do not have compile time knowledge of the type of the object they are implementing [22]. DII, DSI, and other CORBA services, like the Trading and Naming Services [21], offer the basic mechanisms to support a dynamic distributed object environment. In dynamic environments, applications can find out and incorporate new components at run-time. Besides, components can be extended on the fly to incorporate new functionality, and applications can be adapted, also on the fly, to component changes. This level of flexibility is very important to some applications, such as desktops and operating systems [21], network management tools [2], and cooperative applications [12].

Because CORBA allows the discovery of the object type and methods at run time, it is possible to implement mechanisms of *dynamic typing*. The interface repository (IR) offers support for applications to browse object types, and the Naming and Trader services offered by CORBA can be used to address the problem of finding out new objects on the system.

Using DII, the programmer has also access to more method call modes than when using static stubs. CORBA supports three types of calls: *synchronous*, which stands for the traditional RPC semantics, *oneway*, which allows the client to invoke a method and continue its execution without waiting for completion, and *deferred synchronous*, which allows the client to continue its execution immediately after a method call but to later poll the server for a result. This last possibility is available only through the DII.

However, using DII and DSI are not trivial tasks, and involve querying and constructing complex structures. Because C and C++ are static typed languages, with no automatic memory management, a program must build a dynamic call step by step, with explicit calls to create the parameter list, to set each parameter type and value, and so on [13].

It is therefore clear that DII and DSI can be interesting in many cases, but their use represents a difficult task with the current existing support. The next sections present LuaOrb, a binding of Lua to CORBA that offers a more suitable support for developing open applications, and allows the management of change in CORBA based application. LuaOrb interacts with Lua only through the official Lua API, and its implementation required no changes to the language.

3.1 LuaOrb's Client Binding

Lua is a dynamically typed language, wherein all type checking is done at run time. Variables and functions have no type declarations. Objects in Lua (also called *tables*) have no classes; each object can have any kind of methods and instance variables. The mapping between Lua and CORBA has tried to respect this flexibility of Lua. In that way, it should be possible to use CORBA objects in the same way as other Lua objects.

Because CORBA objects were to be accessed like any other Lua object, the generation of stubs was neither necessary nor interesting. Instead, CORBA objects should be accessible from Lua with no need of previous declarations, and with dynamic typing. To achieve this goal, the binding was built upon DII.

LuaOrb uses *proxies* to represent CORBA objects in a Lua program. A proxy is a regular Lua object that uses fallbacks, a reflexive mechanism of Lua, to change its default behavior [7]. When a Lua program calls a method from a proxy, the fallback intercepts the call and redirects it to the LuaOrb binding. Then, LuaOrb dynamically maps parameters types from Lua to IDL, does the actual invocation, and maps any results back to Lua. The mapping of parameter types is done by trying to coerce Lua values into IDL types, and vice versa for result types. This mapping is done between two dynamic type descriptions: the actual types of Lua arguments, accessed through the Lua API, and the formal types of the method parameters, accessed through the Interface Repository.

To illustrate the use of LuaOrb, we will use the following IDL interface:

```
struct book {
  string author;
  string title;
};

interface foo {
  boolean add_book(book abook);
  boolean test();
  long div(long x, long y);
};
```

To create a proxy of a distributed object that implements the `foo` interface, we use the `createproxy` function:

```
a_foo = createproxy("foo")
```

The `createproxy` function has an optional second argument, which is the name of a specific instance of the interface specified in the first parameter. When the second parameter is specified, `createproxy` will only succeed if there is a server object with the same name of this parameter. This function is basically a direct mapping to its equivalent function in the ORB API.

After a proxy has been created, the services of the related object can be requested. For example, the methods of the `foo` interface can be called as follows:

```
a_book = {author="Mr. X", title="New Book"}
a_foo:add_book(a_book)
x = 20
if a_foo:test() then
  x = foo2:div(x,2)
end
```

The first line creates an object with two fields, `author` and `title`, initialized with the given strings. The second line calls the `add_book` method from object `a_foo` (the colon is the Lua operator for method calls). When `a_book` is used as argument in this call, LuaOrb automatically tries to convert it to the IDL structure `book`; because `a_book` has the correct fields with the correct types, the conversion succeeds (otherwise, LuaOrb would signal an error). The conversion works recursively, so a list of Lua tables can be automatically converted to an array of records, for instance.

Because of its dynamic nature, the type conversions allow many changes in an IDL interface not to affect its uses in Lua, such as reordering and removing of structure fields, and changes between IDL types with similar representations in Lua, such as `short` and `long`, or `array` and `sequence`. The type system that emerges from these properties is one in which *structural compatibility* is enforced [19].

As mentioned previously, use of the DII allows a mode of invocation called *deferred synchronous*, where a client triggers a method and later polls for completion. From the server's point of view, it is transparent whether this invocation mode is used or not. To specify a deferred synchronous call in Lua, the programmer simply prefixes the method name with `deferred`; for instance,

```
a_foo:deferred_test()
```

A deferred method call returns a handler that can be used later for retrieving the method result.

Deferred synchronous calls are specially natural in event-driven programming. In this setting, it is interesting to be able to define a function to be called upon the completion of the method. This is supported in LuaOrb through the `completion_event` function, which takes as parameters a handler returned by a deferred synchronous method call and a function. Upon method completion, this function is called with the method results as parameters.

3.2 LuaOrb's Server Binding

The server binding of LuaOrb allows Lua objects to be used as CORBA servers. For that, it uses CORBA's Dynamic Skeleton Interface (DSI). The basic idea of the DSI is to implement all calls to a particular object (which we will call a *dynamic server*) by invocations to a single upcall routine, the *Dynamic Implementation Routine* (DIR). This routine is responsible for unmarshalling the arguments and for dispatching the call to the appropriate code. The most frequent application of DSI has been to implement bridges between different ORBs [23]. In this context, the dynamic server acts as a proxy for an object in some other ORB.

Similarly to the client binding, LuaOrb also uses proxies for the server binding, but reverses the roles: while in the client side each CORBA object has a Lua object as its proxy, in the server side each Lua object that acts as a CORBA server is represented by a proxy, which is a C++ object. This object handles all method requests for the original Lua object through the DSI upcall routine. For each call received, the upcall routine unmarshalls the arguments, converts them to appropriate Lua values, calls the corresponding Lua method, and converts back the results from Lua to CORBA. The conversions follow the same rules used by the client binding, but with opposite directions.

To see a very simple example of a server written in Lua, let us consider the following IDL interface:

```
interface listener {
  oneway void put(long i);
}

interface generator {
  void set_listener(listener r);
}
```

Our `generator` object generates numbers (random numbers, for instance), and for each number it generates it calls the `put` method from its listener. We can write and use a listener object in LuaOrb as follows:

```
l = { put = function (self, i)
              print(i)
            end
    }

gen = createproxy("generator")
gen:set_listener(l)
```

In the first assignment, we create a Lua object with a single method, called `put`. Then, we create a proxy for a generator, and set that Lua object as its listener. When LuaOrb detects that the formal parameter type is an IDL `listener`, while the actual argument is a Lua object, it automatically creates a proxy for this Lua object, enabling it to work as a CORBA server.

LuaOrb also offers an IDL interface for remote server update:

```
interface LuaDSIObject {
  readonly attribute Object obj;
  void InstallImplementation (string opname, string luaCode);
}

interface ServerLua {
  LuaDSIObject Instance (CORBA::InterfaceDef intf);
}
```

To create a new instance at a dynamically extensible server, a client first invokes method `Instance`. The single parameter of this method is a reference to an interface definition in the interface repository. This reference is used by `Instance` to retrieve information about the attributes and methods of the new object.

The new object is returned as the attribute obj inside a new LuaDSIObject (the object in this attribute is, in fact, the proxy for the new Lua object). The manager client can then invoke the method InstallImplementation to install or modify each of the object's methods.

To illustrate the use of this interface, we will change our last example, so that the listener is created remotely, in another machine. We assume that the user has already created bindings to the Interface Repository (IR) and to a remote ServerLua (Server):

```
-- creates a new object with type ‘listener’
l = Server:Instance(IR:lookup("listener"))
-- creates a new method called ‘put’
l:InstallImplementation("put",
                        "function (self, i) print(i) end")
gen = createproxy("generator")
gen:set_listener(l.obj)
```

The presented sequence of commands can be interactively issued from a simple LuaOrb console, which thus becomes a powerful tool for interactive dynamic configuration.

3.3 Access to the Interface Repository

The interface repository (IR), defined as a component of the ORB, provides dynamic access to object interfaces. The IR is itself a CORBA object, and can thus be accessed through method invocations. In general, these methods can be used by any program, allowing the user, for instance, to browse through available interfaces. However, the IR is specially important for DII and DSI, the dynamic interfaces of CORBA. DII allows programs to invoke CORBA servers for which they have no precompiled stub. In order to build dynamic invocations, the program must possess information about available methods and their parameters; the interface repository provides this information. On the server side, DSI allows a server to handle requests for which it has no precompiled skeleton. Again, the correct signature of these requests must be obtained from the interface repository.

LuaOrb provides a library, called LuaRep, to simplify access to the IR. LuaRep makes extensive use of Lua data description facilities. Through the use of fallbacks, LuaRep reflects the repository information into Lua objects, so that any operation on these objects is automatically converted to the equivalent operation on the repository. This allows the Lua programmer to manipulate the repository information by accessing regular Lua objects. The reflexivity implemented by LuaRep allows not only queries to the repository, but updates as well.

The possibility of dynamically updating the IR extends the flexibility obtained with LuaOrb. It allows a manager client to install not only new implementations for existing interfaces, but also unforeseen services in the server, by first adding their definitions to the interface repository.

4 Configuration of Applications

In languages like Conic [10] and Darwin [11], the interface of each component is described in terms of input and output channels. In Darwin, for example, a component is described in terms of the communication objects it *provides* (roughly comparable to input channels) and the communication objects it *requires* (roughly comparable to output channels). This approach allows the specification of different paradigms of interaction between processes [1]. For instance, filtering structures can be easily built from components such as the one shown below:

```
component filter {
  provide left <port,int>;
  require right <port,int>;
}
```

In the CORBA model [21], interfaces are described in IDL by method signatures, instead of input and output ports. Method signatures can declare input and output parameters, besides optional return values. Thus, a method signature describes a bidirectional flow of information.

The two approaches work at different levels of abstraction. In Darwin, the interface of a component contains complete information about all the communication in which the component is involved. Part of the communication activity of a component typically relates to offered services, while some of it may relate to a specific implementation. As an example, a component that implements a parser may or may not communicate with a file server, according to the way it stores temporary information. In CORBA, only offered services are declared in the interface of a component, and nothing is known about the methods which will in turn be called by this component for implementation purposes. On one hand, this gives the programmer much less control over the construction of a complete application. On the other hand, this mechanism supports the use of existing services as black boxes, about whose implementation the programmer needs not worry. Rather than coupling service requirements to service provisions, the building of an application in this case consists mainly of decisions about which services to use.

In some cases, however, the need for objects that offer specific methods may be part of a CORBA object interface. In object oriented programming, an important concept is that of a *listener* or a *callback* object. The specification of an object may define that it must pass on produced data to a consumer object or register the occurrence of certain events at a registrar object. In such cases, since communication is part of specification, and not implementation, the interface of the object must reflect the existence of these communication partners, for instance by providing methods for defining who the partner is.

Recently, languages like Darwin and OLAN [3] are being called *architecture description languages*. This reflects the emphasis on a top-down approach, where support is given for the specification of application structure, and tools are offered for the translation of this specification to module skeletons, which must be filled in by the programmer. In component based programming, a bottom-up approach to software development may be more appropriate. The goal here would be, given the need for a new application, to compose it from existing components keeping new code to a minimum. Since the focus is on using existing components, an important role of the configuration language is to act as an intermediate between components not originally designed to be used together.

In our system, the description of the application is a script written in LuaOrb. The possibility of invoking arbitrary CORBA servers with no need for previous declarations allows the set of CORBA components which compose the application to dynamically evolve. This evolution is implicit: There are no explicit linking or unlinking operations.

Lua and the LuaOrb framework provide an environment that allows the application to evolve in yet another way. Through the `ServerLua` interface, a specific component can be modified or extended from a Lua program running on a different machine. This program may be part of the distributed application or may be executing as an independent activity.

In the next subsections, we discuss some classes of change management in CORBA based applications. Other configuration examples are presented in [18].

4.1 Interfaces and Application Configuration

The goal of our first example is to discuss how the concepts of "provisions" and "requirements", which are present in classic configuration languages, can be modeled using Lua and CORBA. The example we use is the primes sieve of Erathostenes discussed for Darwin in [9] and [11]. In this application, a process feeds a stream of numbers to a pipeline of *filter* processes. Each filter prints the first number it receives and subsequently filters out multiples of that number from the stream of numbers.

The interface definition of the filter component used in Regis is basically the one we presented previously. [11] describes two ways in which a pipeline can be created with these components. In a static configuration, a fixed number of filter components is instantiated, and each `right` port is connected to a `left` port of another filter through a `bind` command. In a dynamic configuration, each filter component is dynamically instantiated when its predecessor in the pipeline uses its `left` port. In what follows, we will discuss only the dynamic configuration, since our main focus here is on application reconfiguration.

The IDL for the filter component in CORBA, *primeFilter*, is as follows: The `provider` interface plays the part of an input channel. It gets new items from calls to its single method `put`.

```
interface provider {
  oneway void put(long i);
}
```

The `requirer` interface models an output channel, that is, a component requiring an input channel. Its single method, `setprovider`, connects it to an input channel, so that, for each new item the interface needs to output, it calls the put method of its provider.

```
interface requirer {
  void setprovider(provider r);
}
```

Finally, the primeFilter interface inherits from these two other interfaces. Therefore, it acts as a filter, receiving numbers through its put method, and passing them to its provider.

```
interface primeFilter: provider, requirer {
}
```

The specific behavior of this filter is quite simple: It stores the first number it receives. After that, it filters out all numbers that are multiple of that first number; only non-multiple numbers pass through it. An implementation of primeFilter is sketched in Figure 1.

```
class primeFilter {
  provider prov;
  long myprime;
  primeFilter () { myprime=0; }
  void setprovider (provider n) { prov=n; }
  void put (long c) { if (!myprime) myprime = c;
                      else if (c%myprime) prov.put(c); }
}
```

Fig. 1. Implementation of primeFilter

Figure 2 presents the configuration of the sieve program in LuaOrb. The program prints out primes from 2 to N. The Lua object assigned to variable lnf is responsible for the dynamic creation of new filter components. After the declaration of the lnf object, a generator object is created. This component simply creates a stream of numbers from 2 to N, to be fed to the filter pipeline.

In this example, as in the next, we use an appServer object to create new instances of a component. The appServer object represents an *application server*, a service that registers interface implementations and supports requests both for running implementations and for new instances of a given interface.

When the generator invokes put for the first time, it will activate the Lua object lnf, which is its current provider object. Execution of lnf:put results in the creation of a new primeFilter object, which is then set as the provider for the generator. Subsequent invocations of put in the generator will invoke this new filter object. This new filter object has lnf as its provider component, so

```
-- define listener for creation of new filters
lnf = { put = function (self, p)
                print(p)
                local next = appServer:createCorba("primeFilter")
                next:put(p)
                next:setprovider(self)
                last:setprovider(next)
              end
      }
--  start application
gen = appServer:createCorba("generator")
gen:setprovider(lnf)
gen:createStream(2,N)
```

Fig. 2. Configuring the sieve of primes

the same behavior will be repeated each time a `primeFilter` invokes `put` for the first time.

This example also illustrates how LuaOrb allows Lua and CORBA objects to be manipulated homogeneously. Calls to the `setprovider` method defined in the `requirer` interface receive both kinds of objects as arguments in different moments.

Note that in the proposed scheme, each filter component is aware only of the existence of a `provider` object, to which candidates to primes must be passed on. The Lua script creates these new filters, as it identifies new primes, in a way that is completely transparent to the C++ components.

The paradigm used in this example is data driven. The generator is the main active object; filters act as passive objects, activated when a new datum is available. We could build this same example with a result-driven paradigm, wherein we change the roles of providers and requirers: Providers would have a single method `get`, which returns a new item, and requirers would call this method from their providers whenever they need a new item. The possibility of these two dual approaches illustrates the flexibility offered by CORBA and LuaOrb.

4.2 Event-Driven Applications

An important class of distributed applications are event-driven applications, where actions must be associated to the occurrence of specific events. This section discusses how CORBA-based event-driven applications can be coded in LuaOrb.

To illustrate the proposed programming style, we will get an example used in [17] to present Glish, a language that supports communication through events. In this example, a simple distributed application is composed out of two components: one measures some data and the other displays the data. The monitor component announces it has produced new data by generating an event. The response to this event is to activate the display component. Using Lua and CORBA, we can again resort to passing a listener object as an argument to the monitor; the generation of an event is then modeled by an invocation of this object.

Possible IDL interfaces for these two components are as follows:

```
interface monitor {
  void measure(long intervalsize, supervisor s);
};

interface display {
  void newData(double data);
}
```

As described in Section 3.1, LuaOrb provides deferred invocations, which allow an asynchronous style of programming. The Lua configuration program can invoke `measure` asynchronously; it can then proceed its execution and the Lua supervisor object will be called when needed. The configuration code for this example is as follows:

```
-- supervisor object
s = { newData = function (nd)
                  d::newData(nd)
                end
    }
d = appServer:createCorba("display")
m = appServer:createCorba("monitor")
m:deferred_measure(5,s)
...
```

In this style of component linking, all events pass through the configuration program. As discussed in [17], this increases the cost of communication, but also buys flexibility. Suppose that, depending on the value of the data, there is a need to transform it before passing it on to the display component. This can be trivially programmed in `s::newdata`. According to the kind of transformation that is needed, it could either be done by calling a third component, or in the Lua program itself.

When performance is vital, Glish provides point-to-point links between programs, through an explicit `link` statement. Unlike Glish, LuaOrb does not need extra mechanisms for such links. In LuaOrb, this facility follows directly from the fact that Lua objects and CORBA objects are interchangeable. So, we can pass the display component directly as an argument to function `measure`:

```
m = appServer:createCorba("monitor")
d = appServer:createCorba("display")
m:deferred_measure(5,d)
```

4.3 Component Modification

In the previous section we have discussed how new components can be dynamically added to an application and how the flow of data can be dynamically redirected between components. In this section we discuss changes to a component itself.

For this discussion we do not use an example from the literature, since few works discuss runtime changes to components. Instead, we will use an example drawn from network management. Figure 3 shows IDL definitions for a component that provides TCP-related information. The definitions represent information provided by the SNMP standard management information base, MIB-II [14]. The tcpConnTable attribute contains, at any moment, a description of each current TCP connection. Each entry in this table is a tcpConnEntry structure, which contains the local and remote ports and IP addresses, and the state of the connection.

```
struct tcpConnEntry {
        long   tcpConnState;
        string tcpConnLocalAddress;
        long   tcpConnLocalPort;
        string tcpConnRemAddress;
        long   tcpConnRemPort;
};

typedef sequence<tcpConnEntry> tcpConnTable;

interface tcpGroup {
        readonly attribute tcpConnTable connections;
        tcpConnTable FilterConnections();
        ...
};
```

Fig. 3. IDL for network management component

A common problem in network management is the availability of enormous quantities of raw data that must be processed by the management application. To circumvent this problem, the tcpGroup interface provides a FilterConnections method, which returns only the connection table entries that satisfy some criteria.

Using LuaOrb, this interface can be implemented by a component that can be modified dynamically, allowing the administrator to establish different filtering criteria with no need for re-compilation. For instance, first we can create this component with a trivial filter, that returns the whole table of TCP connections:

```
trivialf = "function () return connections end"
```

```
dsiobj = serverlua:Instance(IR:lookup("tcpGroup"))
dsiobj:InstallImplementation("FilterConnections", trivialf)
```

Later, we can redefine this implementation, to select only connections that use local port 79, say:

```
newf = [[
  function ()
    filtertab = {}
    i = 0
    while connections[i] do
      if connections[i].tcpConnLocalPort==79 then
        tinsert(filtertab, connections[i])
      end
      i = i+1
    end
    return filtertab
  end
]]

dsiobj:InstallImplementation("FilterConnections", newf)
```

The `newf` variable holds a string with the code of the new filter function. (Lua uses the double square brackets to delimit literal strings that span several lines.) After installing this new implementation, we can use it to retrieve the table of connections:

```
tcpG = dsiobj.obj
interestingConnections = tcpG:FilterConnections()
```

5 Final Remarks

It seems natural here to relate the problem of component-based configuration with the more general problem of software configuration, which deals with customizing software to different needs and environments.

The use of a textual configuration file for software customization reflects a declarative, static style, and can be compared to the use of a declarative architecture description language in component-based applications. Work in architecture description languages, based on a more declarative linking and unlinking style, has given much focus to the problem of checking configuration consistency [6,4] Parsers that check required properties can also be easily built for configuration files.

This is in fact only another application of the properties of static checking. As is always the case, a static solution represents robustness, avoiding many run time errors, but limits the flexibility that can be provided.

With the use of an extension language for software configuration, the configuration file becomes a program, with the associated difficulties for automatic checkers, but also with the flexibility that only a programming language can provide. Besides, execution errors due to eventual inconsistencies can many times be captured and handled in an interpreted language.

In this work, we proposed extending the use of extension languages to the configuration of component-based applications. No explicit linking or unlinking operations are provided: On the one hand, this means no consistency verifications are possible; on the other hand, it means that the evolution of the application is controlled with a full programming language. As shown in the examples discussed in this work, this results in a great deal of flexibility, allowing not only different patterns of component interaction to be dynamically defined but the components themselves to be easily modified. These facilities are specially important if component-based programming is to fulfill its promise in the fields of software reuse and rapid prototyping.

References

1. G. Andrews. Paradigms for process interaction in distributed programs. *Computing Surveys*, 23(1), 1991. 103
2. Bela Ban. *A Generic Management Model for CORBA, CMIP and SNMP.* PhD thesis, University of Zurich, 1997. 98
3. L. Bellisard, F. Boyer, Riveill, and J. Vion-Dury. Systems services for distributed application configuration. In *Proceedings of the Fourth International Conference on Configurable Distributed Systems*, pages 53–60, Annapolis, MD, 1998. IEEE. 103
4. P. Feiler and J. Li. Consistency in dynamic reconfiguration. In *International Conference on Configurable Distributed Systems*, pages 189–196, Annapolis, MD, 1998. IEEE. 109
5. Luiz H. Figueiredo, Roberto Ierusalimschy, and Waldemar Celes. Lua: An extensible embedded language. *Dr. Dobb's Journal*, 21(12):26–33, December 1996. 97
6. K. Goudarzi and J. Kramer. Maintaining node consistency in the face of dynamic change. In *Third International Conference on Configurable Distributed Systems*, pages 62–69, Annapolis, MD, 1996. IEEE. 109
7. R. Ierusalimschy, R. Cerqueira, and N. Rodriguez. Using reflexivity to interface with CORBA. In *International Conference on Computer Languages 1998*, pages 39–46, Chicago, IL, 1998. IEEE, IEEE. 99
8. R. Ierusalimschy, L. Figueiredo, and W. Celes. Lua – an extensible extension language. *Software: Practice and Experience*, 26(6), 1996. 96, 97
9. J. Magee, N. Dulay, and J. Kramer. Structuring parallel and distributed programs. *IEE Software Engineering Journal*, 8(2):73–82, 1993. 104
10. J. Magee, J. Kramer, and M. Sloman. Constructing distributed systems in Conic. *IEEE Trans. on Software Engineering*, SE-15(6), 1989. 103
11. Jeff Magee, Naranker Dulay, and Jeff Kramer. A constructive development environment for parallel and distributed programs. *IEE/IOP/BCS Distributed Systems Engineering*, 1(5), 1994. 103, 104, 104
12. Philippe Marle, Christoph Gransart, and Jean-Marc Geib. CorbaScript and CorbaWeb: A generic object-oriented dynamic environment upon CORBA. Technical report, Universite de Lille, 1996. 98

13. Marco C. Martins, Noemi Rodriguez, and Roberto Ierusalimschy. Dynamic extension of CORBA servers. In *Euro-Par '99*, Toulouse, France, 1999. 98
14. K. McCloghrie and M. Rose. Management information base for network management of TCP/IP-based internets: MIB-II, 1991. RFC 1213. 108
15. Object Management Group, Inc., Framingham, MA. *The Common Object Request Broker Architecture and Specification; Revision 2.0*, jul 1995. 97, 97
16. J. Ousterhout. *Tcl and the Tk Toolkit*. Professional Computing Series. Addison-Wesley, 1994. 96
17. V. Paxson and C. Saltmarsh. Glish: a user-level software bus for loosely-coupled distributed systems. In *1993 Winter USENIX Technical Conference*, 1993. 106, 107
18. N. Rodriguez, R. Ierusalimschy, and R. Cerqueira. Dynamic configuration with corba components. In *International Conference on Configurable Distributed Systems*, pages 27–34, Annapolis, MD, 1998. IEEE. 104
19. N. Rodriguez, R. Ierusalimschy, and J. L. Rangel. Types in School. *Sigplan Notices*, 28(8), 1993. 100
20. D. Schmidt. Corba products, 1999. `http://www.cs.wustl.edu/ schmidt/corba-products.html.` 95
21. Jon Siegel. *CORBA Fundamentals and Programming*. John Wiley & Sons, 1996. 97, 98, 98, 103
22. Peter Wayner. Objects on the March. *Byte*, january 1994. 98
23. N. M. S. Zuquello and E. R. M. Madeira. A mechanism to provide interoperability between ORBs with relocation transparency. In *Proc IEEE Third International Symposium on Autonomous Decentralized Systems (ISADS'97)*, pages 53–60, Berlin, Germany, 1997. 100

Fast, Error Correcting Parser Combinators: A Short Tutorial

S. Doaitse Swierstra[1] and Pablo R. Azero Alcocer[1,2]

[1] Department of Computer Science, Utrecht University,
P. O. Box 80.089, 3508 TB Utrecht, The Netherlands
{doaitse,pablo}@cs.uu.nl
[2] Univeridad Major de San Simon, Cochabamba, Bolivia

1 Introduction

Compiler writers have always heavily relied on tools: parser generators for generating parsers out of context free grammars, attribute grammar systems for generating semantic analyzers out of attribute grammars, and systems for generating code generators out of descriptions of machine architectures. Since designing such special formalisms and constructing such tools deals with one of the most important issues in computer science, courses on compiler construction have always formed part of the core computer science curriculum.

One of the aspects that make modern functional languages like Haskell [3] and ML [4] so attractive is their advanced type system. Polymorphism and type classes make it possible to express many concepts in the language itself, instead of having to resort to a special formalism, and generating programs out of this. It should come as no surprise that, with the increased expressibility provided by the new type systems, the question arises to what extent such tools may be replaced by so-called combinator libraries. In this paper we will present a combinator library that may be used to replace conventional parser generators.

We will be the first to admit that many existing special purpose tools do a great job, and that our library falls short of performing in an equally satisfying way. On the other hand there are good arguments to further pursue this combinator based route. We will come back to this after we have introduced conventional combinator based parsing, since at that point we will have some material to demonstrate the points we want to make. Let it suffice to say here that the size of our tool boxes is only a fraction of the code size of conventional tools; we will present a library of parsing combinators that takes just less than a 100 lines, whereas it provides many features that conventional tools lack. This paper contains enough room to include the full text of the library; something that can definitely not be said about almost any other existing tool.

Parser combinators have been a joy for functional programmers to use, to study, and to create their own versions of [2,1,5]. In many courses on functional programming they form the culminating point, in which the teacher impresses his students with an unbelievably short description of how to write parsers in a functional language. We will do so here too. Next we will explain the advantages of programming with such combinators, and at the same time present

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 112–131, 1999.

```
module BasicCombinators where
infixl 3 <|>
infixl 4 <*>

type Parser s a = [s] -> [(a,[s])]

pSucceed v input = [ (v    , input)]
pFail      input = [               ]

pSym  :: Eq  s  => s                              -> Parser s s
(<|>) :: Eq  s  => Parser s a        -> Parser s a -> Parser s a
(<*>) :: Eq  s  => Parser s (b -> a) -> Parser s b -> Parser s a

pSym a  (b:rest) = if a == b then [(b,rest)] else []
pSym a  []       = []

(p <|> q) input = p input ++ q input

(p <*> q) input = [ (pv qv, rest  )
                  | (pv   , qinput) <- p input
                  , (qv   , rest  ) <- q qinput
                  ]
```

Listing 1: BasicCombinators0

some extensions. We will however also indicate some of the disadvantages of this extremely simplistic approach; thus in Section 7 we will show how to cure these disadvantages. Finally we will sketch how we have constructed a slightly more elaborate version of our combinators, which are usually considerably faster and are of production quality.

2 Basic Combinator Parsing

In Listing 1 we provide the full text of a library for building parsers. A `Parser s a` is a function (`->`) that takes a sequence of tokens as input (`[s]`) and returns a list (`[ ... ]`), containing all possible ways in which a prefix of the input can be recognized and converted into a value of type `a`, each tupled with the corresponding remaining part of the input (`(a, [s])`). A parser `pSucceed v` recognizes the empty string and returns the value `v` as the witness of this success, whereas the parser `pFail` always fails, and thus returns an empty list of solutions. The function `pSym` takes a single token as parameter and returns a parser that either recognizes just this token or fails. The function `pSym` is thus not a parser but a function that builds a parser. The parser combinator `<|>` constructs a new parser out of two alternative parsers, and thus corresponds to the symbol `|` in context free grammars; all it has to do is simply to apply both parsers to the input and to concatenate the two lists of found parses. The

```
module TreeParsing1 where
import BasicCombinators0

data Tree = Leaf Char
          | Bin Tree Tree
          deriving Show

pTree1 =         pSucceed Leaf
             <*> pDigit1
        <|>      pSucceed (\ _ left right _ -> Bin left  right)
             <*> pSym ’(’  <*> pTree1 <*> pTree1 <*> pSym ’)’

pDigit1 = foldr (<|>) pFail (map pSym "0123456789")

foldr op e (a:x) = a ‘op‘ foldr op e x
foldr op e []    = e
```

Listing 2: TreeParsing1

combinator `<*>`, which takes care of the sequential composition, requires a bit more explanation. The construct used in its definition is called list comprehension. First the parser `p` is applied to the input, and for all possible ways (`<-`) in which this parser `p` succeeds with value `pv` and remaining input `qinput`, parser `q` is applied to this remaining input `qinput`. This in its turn results in a list of `(qv, rest)` pairs. There is now some freedom in combining a result `pv` with its corresponding `qv` values. We have decided here to use function application: the parser `p` has to return a function that is applied to the result of the parser `q`. This fact is clearly expressed in the type of the combinator `<*>`.

In Listing 2 we use the combinators for defining a parser `pTree1` for binary trees that have a digit at their leafs. By loading this module into the Haskell interpreter Hugs we may now call our first parser:

```
TreeParsing1> pTree1 "(2(34))"
[(Bin (Leaf ’2’) (Bin (Leaf ’3’) (Leaf ’4’)),"")]
```

You can see that there is only one way of making the input into a `Tree` and that all input was consumed in doing so (`""`).

You may not immediately see how the second alternative of `pTree1` works. For this we have to carefully inspect the fixity declaration of `<*>`, i.e. the text `infixl 4 <*>`; this defines `<*>` to be left-associative and thus something of the form `f <*> a <*> b <*> c <*> d` is interpreted as `((((f <*> a) <*> b) <*> c) <*> d)`. In our case

```
 pSucceed (\e _ left right _ -> Bin left right)
```

always returns a function that takes 4 arguments, and that is precisely the number of components recognized by the remaining part of this alternative.

3 Extending the Library

We will now exploit an important property of our approach: the combinators are not operators in some special formalism, but instead are just functions defined in a general purpose programming language. This implies that we can write functions returning parsers and pass parsers as argument to other functions. This enables us to introduce a wealth of derived combinators, that take care of often occurring patterns in writing parsers. We have already seen a small example of such new possibilities when we defined the parser that recognizes a single digit. Instead of writing down a whole lot of parsers for the individual digits and explicitly combining these, we have taken the sequence of digits "0123456789", have converted each element of that sequence (map) into a sequence of parsers by applying pSym to it, and finally have combined all these parsers into a single one by applying foldr (<|>) pFail. This function foldr builds a result by starting off with the unit element pFail and then combining this with all elements in the list using the binary function (<|>). Since this is an often occurring pattern the real functional programmer immediately sees an opportunity for abstraction here:

```
pAnyOf  :: Eq s => [s] -> Parser s s
pAnyOf   =  foldr (<|>) pFail . map pSym
pDigit2 =  pAnyOf "0123456789"
```

In Listing 3 we have given definitions for some more often occurring situations, and our tree parser might, using these new combinators, also be defined as:

```
pTree2 =             Leaf <$> pDigit2
       <|> pParens (Bin  <$> pTree2 <*> pTree2)
```

This parser definition now has become almost isomorphic to the data type definition. It should be clear from this example that there is now no limit to extending this library.

4 Advantages of this Approach

As a final example of what can be done we will now show how to construct parsers dynamically by writing a parser for an expression language with infix operators. An example input is:

```
(L+R*)a+b*(c+d)
```

and the code we want to generate is:

```
abcd+*+
```

which is the reversed Polish notation of the input expressions.

The text (L+R*) indicates that + is left (L) associative and has lower priority than *, which is right (R) associative. In this way an unlimited number of operators may be specified, with relative priorities depending on their position in this list.

```
module ExtendedCombinators where
import BasicCombinators
infixl 4 <$>, <$, <*, *>, <**>, <??>
infixl 2 ‘opt‘

pAnyOf :: Eq s => [s]                                -> Parser s s
opt    :: Eq s => Parser s a  -> a                   -> Parser s a
(<$>)  :: Eq s => (b -> a)    -> Parser s b          -> Parser s a
(<$  ) :: Eq s => a           -> Parser s b          -> Parser s a
(<*  ) :: Eq s => Parser s a  -> Parser s b          -> Parser s a
(  *>) :: Eq s => Parser s a  -> Parser s b          -> Parser s b
(<**>) :: Eq s => Parser s b  -> Parser s (b->a)  -> Parser s a
(<??>) :: Eq s => Parser s b  -> Parser s (b->b)  -> Parser s b

pAnyOf    =  foldr (<|>) pFail . map pSym
p ‘opt‘ v = p <|> pSucceed v
f <$> p   = pSucceed f     <*> p
f <$  p   = const f        <$> p
p <*  q   = (\ x _ -> x)   <$> p <*> q
p *>  q   = (\ _ x -> x)   <$> p <*> q
p <**> q  = (\ x f -> f x) <$> p <*> q
p <??> q  =                    p <**> (q ‘opt‘ id)

pFoldr         alg@(op,e)     p
  = pfm where pfm = (op <$> p <*> pfm)   ‘opt‘ e
pFoldrSep      alg@(op,e) sep p
  = (op <$> p <*> pFoldr alg (sep *> p)) ‘opt‘ e
pFoldrPrefixed alg@(op,e) c   p   = pFoldr alg (c *> p)

pList           p = pFoldr         ((:), [])   p
pListSep      s p = pFoldrSep      ((:), []) s p
pListPrefixed c p = pFoldrPrefixed ((:), []) c p

pSome p      =  (:) <$> p <*> pList p
pChainr op x =  r where r = x <**> (flip <$> op <*> r ‘opt‘ id)
pChainl op x =  f <$> x <*> pList (flip <$> op <*> x)
               where
                 f x [] = x
                 f x (func:rest) = f (func x) rest

pPacked l r x   =   l *>  x <*   r

-- some ad hoc extensions
pOParen = pSym ’(’
pCParen = pSym ’)’
pParens = pPacked pOParen pCParen
```

Listing 3: ExtendedCombinators

We start by defining a function that parses a single character identifier and returns as its result that identifier in the form of a string:

```
pVar = (\c -> [c]) <$> pAnyOf ['a'..'z']   .
```

The next step is to define a function that, given the name of an operator, recognizes that operator and as *a result returns a function that will concatenate the two arguments of that operator and postfix it with the name of the operator*, thus building the reversed Polish notation:

```
pOp name = (\ left right -> left++right++[name]) <$ pSym name
```

Note that, by using the operator `<$` we indicate that we are not interested in the recognized operator; we already know what this is since it was passed as a parameter.

Next we define the function `compile`. For this we introduce a new combinator `<@>`, that takes as its left hand side operand a parser constructor `f` and as its right hand side operand a parser `g`. The results `v` of parsing a prefix of the input with `g`, are used in calling `f`; these calls, in their turn, result in new parsers which are applied to the `rest` of the input:

```
(f <@> g) input  = [ f v rest | (v, rest) <- g input ]
```

Since our input consists of two parts, the priority declarations and the expression itself , we postulate that the function compile reads:

```
compile = pRoot <@> pPrios
```

First we focus on the function `pRoot`, that should take as argument the result of recognizing the priorities. Here we will assume that this result is a function that, given how to parse an operand, parses an expression constructed out of operands and the defined operators:

```
pRoot prios = let pExpr = prios (pVar <|> pParens pExpr) in  pExpr
```

There is a difference between an operator that occurs in the declaration part of the input and one in the expression part: the former may be any operator, whereas the latter can only be an operator that has been declared before. For the priority declaration part we thus introduce a new parser that recognizes any operator, and returns a parser that compiles the just recognized operator using the function `pOp` defined before:

```
pAnyOp = pOp <$> pAnyOf "+*/-^"       -- just some possible operators
```

Now suppose we have recognized a left and a right associative operator resulting in operator compilers `pLeft` and `pRight`. Out of these we can construct a function that, given the operand parser, parses infix expressions containing `pLeft` and `pRight` occurrences:

```
pLR factor = (pChainl pLeft . pChainr pRight) factor.
```

Generalizing this pattern to an unlimited number of operators we now deduce the definition:

```
pPrios  =  pParens $
           pFoldr ((.), id)  ((   pChainl <$ pSym 'L'
                              <|> pChainr <$ pSym 'R')  <*> pAnyOp)
```

Let us now compare once more this approach with the situation where we would have used a special parser generator. In the combinator approach we can freely introduce all kinds of abbreviations by defining new combinators in terms of existing ones; furthermore we may define higher order combinators that take arguments and return values that may be parsers. This is a property we get for free here, and is absent in most tools, where the syntax of the input is fixed and at most some form of macro processing is available as an abstraction mechanism.

Another important consequence from embedding our parser construction in an existing language is that type checking and error reporting can directly be done at the program level, and not at the level of some generated program.

5 Analysis

Before going on let us reflect for a while on why this all works so neatly; somehow we managed to define a new language within an existing one. There are many important aspects to be distinguished here.

In the first place having *polymorphic types* is essential. We managed to keep the types of the parser and the types of the result completely separated. There is no way in which the parsers can inspect those values or mutilate them. All they know is that they have types like `a` and `a -> b`. If these are the only things known, the only thing a parser can do is apply the one to the other, but that was the intention of providing these types. So the combinators really are a conservative extension of the rest of the program.

In the second place the *type classes* make it possible to precisely define the interfaces between the parsing part and the rest of the program. For the parsing it is necessary to know whether tokens are equal or not, and precisely this fact is thus specified in the context part (`Eq s =>`) of the type of the combinators. This is the piece of program text needed, but also the only available property of tokens in the parsing part of your programs.

The third issue that makes things look nice is of a more syntactic nature: by being able to define new infix operators, parser definitions can be made to resemble grammars, thus taking away another reason for having a special formalism.

Although the combinators defined before look very attractive, they have some serious shortcomings, that make them almost unusable in practice.

Because we have used the list of successes approach, the result for incorrect inputs will be an empty list, and no indication what went wrong and where is given. Furthermore we rely on backtracking for finding all possible parses; our (extended) combinators were cleverly defined in such a way that they return the longest possible parse first, and this is usually what one wants. When the input contains errors or we are not primarily interested in the greedy solution, there is a high backtracking overhead to be paid.

In the next section we will attack these two problems, and will come up with a set of basic combinators that not only report errors but also repair the input. We will sketch the implementation of an equivalent set of combinators that do not even suffer from the backtracking overhead. Of course there is a price to be paid too: they are far more complicated and the number of lines needed for describing them is almost tenfold; these hundred lines however still compare favorably with the size of equivalent special purpose tools such as YACC which have far less expressive power.

6 Error Locating Parsing Combinators

Before going into techniques for error correction we first spend some time on investigating how to get some form of error reporting. Since the two aspects are deeply intertwined however we will only do so briefly.

In case the input is erroneous we will not be able to return a complete parse, but we may try to report on how far we got in the input. In order to do so we change the type of the parsers such that they not only return a value, but also how many tokens were accepted in the parsing process. To this end an extra argument is tupled with the input: the number of tokens accepted so far. The new combinators are given in Listing 4.

The main disadvantage of this approach is that, once we have discovered where the erroneous point is, we have lost the computation that led us there; this implies that we have both lost the necessary contextual information for deciding what kind of repair to make, and the information to continue with the parsing process from that point on.

This suggests that we do the error correction as soon as we discover an erroneous situation, because then we still have the contextual information at our disposal. This however is not trivial. We may locally discover that we got stuck, but maybe there is some other alternative that will bring us much further; in that case the current state can just be discarded and no further time should be wasted on it. In order to make this decision we have to convert our depth-first backtracking strategy into a breadth-first strategy, in which we work on all possible parses in parallel. Only then will we be able to discover whether correcting actions are worthwhile to be taken, or whether there are still other alternatives present that can make progress without having to perform such corrective actions.

The basic step we take now is to look at the corrective actions and the decision whether they were needed or not separately: we generate a set of candidates containing all possible parses and all possible corrections, and decide elsewhere which candidate wins. This approach may look horrendously expensive, but we will exploit lazy evaluation to prevent the full computation of all these (possibly corrected) parses. This will also take care of another problem: once we start changing the input by adding or deleting symbols, the set of parses is likely to become infinite, and we had better avoid computing this whole collection!

```
type Parser s a = ( Integer, [s]) -> [Result s a]

data Result s a  = Until Integer
                 | Succeed a  (Integer, [s])
                 deriving Show

pSucceed v (n, input) = [Succeed v (n, input)]
pFail      (n, input) = [Until n]

pSym a (n, (b:rest)) = [if a == b then Succeed b (n+1,rest) else
                                                             Until n]
pSym a (n, []      ) = [                                     Until n]

(p <|> q) ninput = p ninput ++ q ninput

(p <*> q) ninput = let presult = p ninput
                   in  [ Until np | Until np <- presult]
                       ++
                       [ Succeed (pv qv) nqrest
                       | Succeed pv nprest <- presult
                       , Succeed qv nqrest <- q nprest
                       ]
                       ++
                       [ Until nq
                       | Succeed pv nprest <- presult
                       , Until nq          <- q nprest
                       ]

parse p input = foldr1 best (p (0,input))
                where a ‘best‘ b = if pos a > pos b then a else b
                      pos (Until n)         = n
                      pos (Succeed _ (n,_)) = n
```

Listing 4: BasicCombinators1

7 Error Correcting Parsing Combinators

Since we have decided to deal with the error correction as an integrated part of the parsing process, we will start with a closer inspection of the kind of corrections we want to make. The point where we discover that no progress can be made is in the function `pSym`, where we expect to see a specific token at the head of the input stream, but unfortunately find something different. In this case there are basically two ways to make progress:

- *insert* the expected token at this position
- *delete* the unexpected token at this position and try again

So a first attempt for the function `pSym` is something of the following form:

```
pSym a input@(b:rest)
   = if a == b then [(b, rest)]
     else [(a, input)]   -- pretend that the token was seen
          ++             --   or
          pSym a rest    -- delete the unexpected token from the input
                         -- and try again
pSym a [] = [(a, [])]    -- pretend the token was there
```

On a close inspection one sees that this version of pSym actually constructs all possible inputs and and tries to match those to the given input.

The question that arises now is which parse to select; the given approach generates a tremendous number of parses, most of them corresponding to heavily mutilated versions of the input. So in our next step in the development we combine each result with information about its quality. For this we introduce a new data type that represents a parsing history as a sequence of acceptance (Okstep) and correction (Failstep) steps. A first attempt of this approach is given in Listing 5. Instead of passing around an integer indicating how many steps were successfully taken in the past, each result is now tupled with how many steps were successfully taken in its recognition: so “counting” starts at the end of the recognized part, instead of at the beginning. For correct inputs the length of the list of steps will, once we return the value at the root, be the same as the integer tupled with the result in the previous attempt.

We have not given here the function best yet, since this solution is erroneous anyway. A moment of thought will show that the final result of the parsing process is, since most languages are infinite, likely to be infinite too: the given input can, with a sufficient number of correcting actions, be changed into each of the sentences of the language. A further shortcoming of this approach is that after recognizing a p<*>q, each sequence of steps of p is prefixed to many q-steps. As a consequence many resulting sequences will have long common prefixes, which makes the comparison process prohibitively expensive. Also the blunt way of concatenating the steps is extremely costly, since building sequences by repeated concatenation of parts tends to be quadratic in the length of the list. It is tempting to first select the best element from the different q-solutions, and only append that solution to the corresponding p-solution, but that is wrong too: a short q-solution may lead to a better starting point for a parser that is invoked after p<*>q. Finally we are likely not to get a result at all, since before we are able to construct part of a result of the form psteps++qsteps we have to find an appropriate qsteps. As soon as we get in an infinite branch of the construction process no more solutions will become available!

8 Continuation Based Parser Combinators

A new insight has now popped up, however. If we could, after recognizing a p<*>q-structure, peek into the future to see what the consequences are of taking specific alternatives, we could report back to our caller about its future by combining our local information with that information about our future. Experienced

```
module BasicCombinators2 where
infixl 3 <|>
infixl 4 <*>

data Step  = Okstep
           | Failstep
type Steps = [Step]

failsalways = Failstep:failsalways

type Parser s a = [s] -> [(a, Steps, [s])]

pSucceed v input = [(v        , []         , input)]
pFail      input = [(undefined, failsalways, input)]

pSym a input@(b:rest)
  = if a == b
    then [(b, [Okstep], rest)]
    else [(a, [Failstep], input)]
                     ++
         [(v, Failstep:steps, r) | (v, steps, r) <- pSym a rest]

pSym a [] = [(a, [Failstep], [])]

(p <|> q) input = p input ++ q input

(p <*> q) input = [ (pv qv, psteps++qsteps, rest)
                  | (pv   , psteps        , qinput) <- p input
                  , (qv   , qsteps        , rest  ) <- q qinput
                  ]

parse p input = foldr1 best (p input)
```

Listing 5: BasicCombinators2

functional programmers will smell the use of continuation based techniques here. So the question that now arises is: What should be our new `Parser`-type?

Before answering this question let us look for a moment at the data structures used in a conventional description of a top-down parser for context free languages:

- The stack of the symbols recognized thus far, capturing the history of the parsing process and to be used in the construction of the final result.
- The state of the parser, consisting of a stack of symbols still to be recognized.
- The unconsumed part of the input.

In a continuation passing style all such data has to be passed around on by means of parameters. So a parser that should recognize something of type `a`, takes the following arguments:

- a *history*, that may be combined with the recognized value of type a into a new history of type b, that is to be passed on to
- a *future* that will eventually construct something of type d, when passed
- the *remaining input*

So our parser should be of the following type:

```
Future s b d -> History a b -> [s] -> Result d
```

with appropriate definitions for Future, History and Result. The obvious choice for the history is

```
type History a b = a -> b
```

because this is the simplest type that can hold values that convert a's into b's. The type for the future does not leave us much choice either, since it has to accept the newly constructed history of type b and the remaining input of type [s] and should produce something tat contains a type d value:

```
type Future s b d = b -> [s] -> Result d
```

We might have taken the liberty to let the future return a value of type Result' s d, but that does not turn out to be necessary.

Finally let us try to design the type Result d. A parser gets this value back from the future (i.e. the called continuation), and has to return it on to its past (i.e. its caller): the type Result in Listing 6, has been designed in such a way that it both represents the final result and the parsing steps in finding that result. The Cost field will, for the time being, be chosen to be 0 when an input token was successfully recognized, and 1 whenever a symbol was inserted or deleted.

Note that we do not use a conventional continuation passing style, in which continuation calls are usually so-called tail-calls, in which the result of the continuation call becomes the result of the calling function. Here we take the opportunity to add some information to the result, before returning it to our own caller: i.e. we add information about the parsing steps that were taken between being called and calling our continuation.

In Listing 6 we present the final solution, and we will go through this solution step by step. In lines 1–11 we repeat the types introduced thus far. The type Parser is a bit peculiar since it contains two type variables that do not occur as a parameter. In many extensions of Haskell98 however it is possible to denote such universally quantified types, provided we locate them inside a newtype definition; here such a newtype definition is for all practical purposes equivalent to a normal type definition, with the exception that it introduces an extra constructor (P).

The function definition of pSucceed is straightforward: it combines its history h with the witness of its success (v) into a new history (h v), and passes this, together with the input on to its own future f. The function pFail simply returns an infinite list of fail steps.

The sequential composition p<*>q starts the parsing of the composition of p and q by calling p. Since after p first q and then their common future f should be parsed, we construct the future for p by partially applying q to f. The history of the call to p should be such that when it is applied to pv its result is a function,

```
newtype Parser s a  = P (forall b, d
                        .   Future s b d
                         -> History a b
                         -> [s]
                         -> Result d
                        )
type Future s b d = b -> [s] -> Result d
type History  a b = a -> b
type Cost         = Int
data Result d     = Step Cost (Result d)
                  | Stop d
-- THE PARSER COMBINATORS
pSucceed v      = P (\ f h input -> f (h v)                  input  )
pFail           = P (\ f h input -> fails where fails = Step 1 fails)
(P p) <*> (P q) = P (\ f h input -> p (q f) (h .)            input  )
(P p) <|> (P q) = P (\ f h input -> p f h input ‘best‘ q f h input  )

pSym a     = P  (
 \ f h -> let pr = \ input ->
                   case input
                   of  (s:ss)  ->
                         if s == a then Step 0 (f (h s) ss)
                                   else Step 1 (pr      ss)  -- delete
                                               ‘best‘
                                      Step 1 (f (h a) input) -- insert
                       []      ->     Step 1 (f (h a) input) -- insert
          in pr )

-- SELECTING THE BEST RESULT
best :: Result v -> Result v -> Result v
left@(Step l aa) ‘best‘ right@(Step r bb) = if      l < r then left
                                            else if l > r then right
                                            else Step l (aa ‘best‘ bb)
(Stop v   ) ‘best‘ _            = Stop v
_           ‘best‘ (Stop v)     = Stop v
```

Listing 6: BasicCombinators3

that when applied to `qv` results in `h (pv qv)`, and we see that the function `(h .)` does the job since `h (pv qv) == (h . pv) qv == (h .) pv qv`. For `p<|>q` we simply call both alternatives with the same history and future and choose the `best` result of the two.

The function `pSym` checks the first symbol of the input; if this is the sought symbol the continuation `f` is called with the new history `(h s)` and the rest of the input `ss`. Once this returns a result, the fact that this was a successful parsing step is recorded by applying `Step 0` to the final result, and that value is returned to the caller. If the sought symbol is not present both possible corrections are performed. Of course, when we have reached the end of the input, the only possible action is to try to insert the expected symbol.

We have kept the most subtle point for desert, and that is the definition of the function `best`. Its arguments are two lists with information about future parsing steps, and ideally it should select the one containing the fewest corrections. Since computing this optimal solution implies computing all possible futures that at least extend to the end of the input, this will be very expensive. So we choose to approximate this with a greedy algorithm that selects that list that is better than its competitor at the earliest possible point (i.e. that list that comes first in a lexicographic ordering). We should be extremely careful however, since it may easily be the case that both lists start with an infinite number of failing (`Step 1`) steps, in which case it will take forever before we see a difference between the two. The function `best` has carefully been formulated such that, even when a final decision has not been taken yet, already part of the result is being returned! In this way we are able to do the comparison and the selection on an incremental basis. We just return the common prefix for as far as needed, but postpone the decision about what branch is to be preferred, and thus what value is to be returned, as long as possible. Since this partial result will most likely again be compared with other lists, most such lists will be discarded before the full comparison has been made and a decision has been taken! It is this lazy formulation of the function `best` that converts the underlying depth-first backtracking algorithm into one that works on all possible alternatives in parallel: all the calls to `best`, and the demand driven production of partial results, drive the overall computation. In the next section we will complete our discussion by describing how to call our parsers, and what functions to pass to our initial parsers.

9 Further Details

Having constructed a basic version of the parser combinators, there still is some opportunity for further polishing. In this section we will describe how error reporting may be added and how constructed parsers have to be called. We finish by pointing out some subtle points where the innocent user might be surprised.

9.1 Error Reporting

Despite the fact that we have managed to parse and correct erroneous inputs, we do not know yet, even though we get a result, whether any or what corrections were made to the input. To this end we now slightly extend the parser type once more as show in Listing 7. All parsing functions take one extra argument, representing the corrections made in constructing its corresponding history. When further corrections are made this fact is added to the current list of errors. Errors are represented as a value of type `Errors s -> Errors s`, so actually we are passing a function that may be used to construct the `Errors s`

```
newtype Parser s a
 = P (forall b, d.   Future s b d
                  -> History a b
                  -> Errs s
                  -> [s]
                  -> Result s d
     )

type Errs s = (Errors s -> Errors s)

data Errors s = Deleted  s s        (Errors s)
              | Inserted s s        (Errors s)
              | InsertedBeforeEof s (Errors s)
              | DeletedBeforeEof  s (Errors s)
              | NotUsed [s]

instance Show s => Show (Errors s) where
  show (Deleted  s w e       ) = msg "deleted "  s (show w)        e
  show (Inserted s w e       ) = msg "inserted " s (show w)        e
  show (InsertedBeforeEof s e) = msg "inserted " s "(virtual) eof" e
  show (DeletedBeforeEof  s e) = msg "deleted "  s "(virtual) eof" e
  show (NotUsed []           ) = ""
  show (NotUsed ss           ) = "\nsymbols starting with "
                                 ++ show (head ss)
                                 ++ " were discarded "

msg txt sym pos resterrors = "\n" ++ txt ++ show sym
                             ++ " before " ++ pos ++ show resterrors

-- the new version of pSym
pSym a f h
  = P( let pr =
        = \ errs input
          -> case input
             of  (s:ss)  ->
                  if s == a
                  then Step 0 (f (h s) errs ss  )
                  else Step 1 (pr (errs . (if null ss
                                           then DeletedBeforeEof s
                                           else Deleted s (head ss)
                                  )       )
                                  ss
                              )
                                            `best`
                    Step 1 (f (h a) (errs .Inserted a s       ) input)
              [] -> Step 1 (f (h a) (errs .InsertedBeforeEof a) input)
       in pr )
```

Listing 7: ErrorReporting

that we are interested in. As a consequence all combinators change a bit, in the sense that they all get an extra argument that is just passed on to the called continuations. Only when corrections are applied in order to be able to continue we have to do something, and we have seen that this is local to the function pSym. The new version of pSym is given in Listing 7.

The errors are returned in the form of a data type, that may be converted into a string using the function show.

9.2 How to Stop?

Before we can actually parse something we still have to decide what kind of continuation to pass to the parser corresponding to the root symbol of the grammar. We could have defined this function once and for all, but we have decided to provide some extra flexibility here. In our expression example we have already seen that one may not only want to stop parsing at the end of the input. The function parse takes a Boolean function that indicates whether parsing has reached a point that may be interpreted as the end of the input. If this is the case then no error message is generated. Otherwise it is reported that there were unconsumed tokens (which are assumed to have been deleted). Furthermore not only the witness of the parsing is stored in the resulting value, but also the accumulated errors and the remaining part of the input.

```
parse (P p) user_eof input
   = let eof v e input
          =  if user_eof input
             then         (Stop (v, input,  e  (NotUsed []   )))
             else foldr   (\ _ t -> Step 1  t)
                          (Stop (v, input,  e  (NotUsed input)))
                          input
         stepsresult (Step _ s)   = stepsresult s
         stepsresult (Stop v)     = v
     in  stepsresult ( p    -- the parser
                       eof  -- its future
                       id   -- its history
                       id   -- no errors thus far
                       input )
```

10 Pitfalls

One of the major shortcomings of programming in the way we do, i.e. with many higher order functions and lazy – and possibly infinite – data structures is that a feeling for what is actually going on and how costly things are, easily gets lost.

The first example of such a pitfall occurs when in the input a simple binary operator is missing between two operands. Usually there are many operators that might be inserted here, all leading to a correct context free sentence. Unfortunately the system is, without any further help, not able to distinguish between all these possible repairs. As a consequence it will parse the rest of the program once for each possibility, frantically trying to discover a difference that will never show up. If several of such repairs occur the overall parsing time explodes. It is for this reason that we have included the cost of a repair step in an Int-value. If this phenomenon shows up, all operators but one can be given a higher insertion cost, and as a consequence one continuation is immediately selected without wasting time on the others. Furthermore, in the complete version of our library[1] the lookahead for the best function is limited somewhat, once it has been discovered that two sequences that both contain a repair step are being compared. It is clear that for the function best there is still a lot of experimentation possible.

A second problem arises if we have a grammar that may need a long lookahead in order to decide between different alternatives. An example of such a grammar is:

```
p =     count <$> pSym 'a' <*> p <*> pSym 'b'
    <|> count <$> pSym 'a' <*> p <*> pSym 'c'
    <|> count <$> pSym 'a' <*> p <*> pSym 'd'
    <|> count <$> pSym 'a' <*> p <*> pSym 'e'
    where count _ n _ = n+1
```

Here some heavy backtracking will take place, once we try to parse an input that starts with many a's. Of course this problem is easily solved here by rewriting the grammar a bit using left-factorization, but one may not always be willing to do so, especially not when semantic functions like count are different. Fortunately it is possible to exchange time for space as we will show in the next section, and this will be done by the combinators from our library.

11 Speeding Up

Despite its elegance, the process of deciding which alternative to take is rather expensive. At every choice point all possible choices are explored and usually all but one are immediately discarded by the calls to best. In most parsers the decision what to do with the next input symbol is a simple table lookup, where the table represents the state the parser is in. Using the technique of tupling we will sketch how in our parsers we may get a similar performance; since the precise construction process is quite complicated, time and space forbid a detailed description here. Furthermore we assume that the reader is familiar with the construction of LR(0) items, as described in every book on compiler construction.

The basic data structure, around which we center our efforts, is the data type Choices in Listing 8. This data structure describes a parser, and is tupled with its corresponding real parser using the function mkParser; this is an extension of the techniques described for LL(1) grammars in [5].

The four alternatives of Choices represent the following four cases:

Found In this case there is no need to inspect any further symbols in the input; the parser P s a should be applied at this point in the input.

[1] http://www.cs.uu.nl/groups/ST/Software/Parse/

```
data Choices s a = Choose (P s a)     [(s,  Choices s a)]
                 | Split  (P   a)          (Choices s a)
                 | End    (P s a)
                 | Found  (P s a)          (Choices s a)

cata_Choices (sem_Choose, sem_Split, sem_End, sem_Found)
  = let r
      = \ c -> case c of
               (Choose p     csr) -> sem_Choose p [ (s, r ch)
                                                  | (s,   ch) <- csr
                                                  ]
               (Split  p cs     ) -> sem_Split  p (r cs)
               (End    p        ) -> sem_End    p
               (Found  p cs     ) -> sem_Found  p (r cs)
    in r

mkparser cs
 =let choices
       = cata_Choices
          (\ (P p) css                                 -- shift
            -> \inp ->  case inp
                        of []     ->  p
                           (s:ss) ->  case find cmp css s of
                                      Just (_, cp)  -> (cp ss)
                                      Nothing       ->  p
          ,\ (P p) css -> \inp -> p ‘bestp‘ (css inp)  -- reduce and
                                                          shift
          ,\ (P p)     -> \_   -> p                    -- reduce
          ,\ (P p) cs  -> \_   -> p                    -- only
                                                          candidate
          ) cs
 in (cs, (P (\ f h e input -> (choices input) f h e input))) -- tuple

p ‘bestp‘ q  = ( \ f h e input -> p f h e input ‘best‘ q f h e input )
```

Listing 8: MakeParser

Choose Based on the look-ahead inspected thus far it is not yet possible to decide which parser to call, so we have to use the `[(s, Choices s a)]` structure to continue the selection process. If the next input symbol however is not a key in this table the corresponding `P s a` is the error correcting parser that applies here. This state corresponds to a pure shift state, in LR(0) terminology.

End This corresponds to a pure reduce state. The parser `P s a` is the parser we have to call, and we can be sure it will succeed since in the selection process we have seen all the symbols of its right-hand side.

Split This corresponds to a shift-reduce state and we have two possibilities. So we continue with the selection process in the `Choices s a` component, and apply both the parser found there and the `P s a` component of the `Split`, and compare the results using the function `best`.

The function `cata_Choices` is a homomorphism over the initial data type `Choices`: it returns a function that replaces each data constructor occurring in its argument with the corresponding function from the argument of `cata_Choices`. The function `mkparser` is defined that tuples a `Choices s a` structure `cs` with a real parser. This demonstrates another important technique that can often be applied when writing functions that can be seen as an interpreter: *partial evaluation*. In our case the "program" corresponds to the choice structure, and the input of the program to the input of the parser. The important step here is the call to `cata_Choices` that maps the choice structure to a function that chooses which parser to call. This resulting function is then used in the actual parser to select the parser that applies at this position (`choices input`), which parser is then called: `(\ f h e input -> (choices input) f h e input)`.

12 Conclusions and Further Reading

We have shown how, by making use of polymorphism, type classes, higher order functions and lazy evaluation we can write small libraries for constructing efficient parsers. In defining parsers with this library all features of a complete programming language are available.

Essential for the description of such libraries is the availability of an advanced type system. In our case we needed the possibility to incorporate universally quantified types in data structures.

We expect that, with more advanced type systems becoming available, special purpose tools will gradually be replaced by combinator libraries.

Acknowledgments

We want to thank all the people who have been working with us in recent years on the problems described. We want to thank especially Sergio de Mello Schneider, David Barton and Oege de Moor for showing interest in the tools we have constructed, for using them and providing feed back. We want to thank Jeroen Fokker and Eelco Visser for commenting on an earlier version of this paper.

References

1. Hutton, G., Meijer, E. Monadic parser combinators. *Journal of Functional Programming*, 8(4):437–444, July 1998. 112

2. Fokker J. Functional parsers. In Jeuring J. and Meijer E., (Eds.), *Advanced Functional Programming*, Vol. 925 in *Lecture Notes in Computer Science*, pp. 1–52. Springer-Verlag, Berlin, 1995. 112
3. Hammond, K., Peterson, J. (Eds). Haskell 1.4 report. Available at: http://www.haskell.org/, May 1997. 112
4. Milder R., Tofte M., Harper R. *The Definition of Standard ML.* MIT Press, 1990. 112
5. Swierstra S. D., Duponcheel L. Deterministic, error-correcting combinator parsers. In John Launchbury, Erik Meijer, and Tim Sheard (Eds.), *Advanced Functional Programming*, Vol. 1129 in *Lecture Notes in Computer Science*, pp. 184–207. Springer-Verlag, 1996. 112, 128

IBM SanFrancisco: Java Based Business Components, and New Tools to Develop Applications

Ghica van Emde Boas

IBM SanFrancisco Support Group,
IBM, Watsonweg 2, 1423 ND Uithoorn, the Netherlands
emdeboas@nl.ibm.com

Abstract. IBM SanFrancisco*[1] provides application developers with a base set of object-oriented infrastructure and application logic components. These components are delivered as a set of frameworks, currently covering the domains of general ledger, warehouse management, order management, accounts receivable, and accounts payable. After a brief overview of the IBM SanFrancisco frameworks, its structure and contents, we will discuss the requirements for new application development tools, and practical research done to show the effectiveness of such tools.

1 Introduction

Components are quickly becoming mainstream in business application development. The internet revolution has forced application development organizations to find more effective ways of development and the use of Components is one way to provide better, more robust applications faster.

The use of components does not make application development easier. The real development effort of using components includes the learning curve of becoming familiar with the functionality of the component and it programming interface. This learning curve can be very steep and should be taken into account when an organization adopts component technology. Another factor easily overlooked is that state-of-the art component frameworks will use Object Technology and will be built in the Java** programming language. Most traditional development organizations need time, typically a year, to adapt to these new programming paradigms.

Appropriate tools will help with application development using components. Tools such as Rational Rose**, are commonly used during analysis and design of object-oriented applications, and IBM's VisualAge* for Java is a prominent integrated development environment (IDE) in this area. These tools are adapting to component frameworks, but are still lacking features. This article surveys which features are needed for effective tools to be used with component frameworks.

[1] Trademark acknowledgements can be found at the end of this article. They are indicated with * and ** in the text.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 132–149, 1999.

We do so by looking at the architecture of one of such frameworks, IBM SanFrancisco, and then derive requirements to support this architecture in application development. Next, we will look at an example of new tool architecture, developed by the author, which is capable to support these requirements in a flexible way.

The IBM SanFrancisco project is a very large component building effort. It is unique because it includes not only middleware, but also a large base of business components. We will give an overview of SanFrancisco in the next section. We start with an overview, more information can be found at [4].

2 Overview of SanFrancisco

The IBM SanFrancisco shareable frameworks is a distributed object infrastructure and a set of application business components. SanFrancisco delivers three layers of reusable code written in Java, for use by application developer. They are:

- Foundation layer
- Common Business Object Layer
- Core Business Process layer

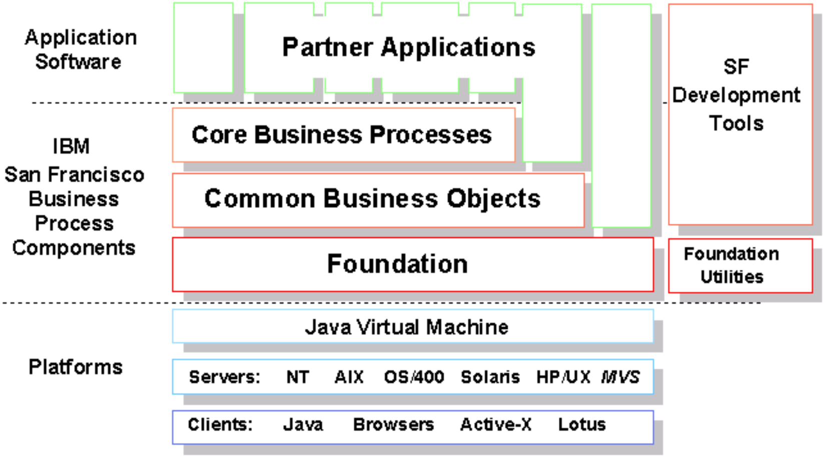


Fig. 1. The structure of SanFrancisco

Fig. 1 shows the layered architecture of SanFrancisco. In addition to the three layers just mentioned, you see Applications, which can be built on top of any of these layers. From an application development point of view, the benefits of using SanFrancisco will depend on which layer is used as a base for the application. The benefits will be highest when the *Common Business Processes* are used as a base. On this level, typically 40–60 % of the functionality of an application is already provided by the framework. It allows the application developer to concentrate on differentiators, not on basic function. In the authors experience, many developers think at first that the SanFrancisco model is too complicated for their purpose. Gradually during development they usually start to appreciate the flexibility of the framework.

From a technology perspective, applications based on shared frameworks, object-oriented technology and Java, provide the best opportunity for interoperability across multiple applications and platforms. Eventually, the customers will be able to purchase applications which are more flexible, allow choice of platforms and allow interoperability with a choice of applications.

2.1 Foundation Layer

The Foundation layer provides the infrastructure and services that are required to build applications in a distributed, multi-platform environment. Its main services are: *Utilities,* for example: Installation, configuration, and administration; and *Kernel services,* for example: Transaction, Persistence, and Security.

The foundation layer also provides a basic object structure, from which all other components in the framework inherit. The major object model base classes are: **Entity** and **Dependent.** Entities are persistent, transactable and distributable. All major components inherit from Entity. Dependents do not have an independent life-cycle. They are more light-weight and are only persistent when owned by an Entity instance.

2.2 Common Business Objects

The Common Business Objects layer provides implementations of commonly used business objects that are common to more than one domain. The Common Business Objects can also be used as a base for interoperability between applications. We enumerate the currently existing common business objects here, to show the breadth of functionality provided:

- *General-purpose business objects:* Company, Business Partner, Address, Initials, Classification, Number series
- *Common business processes:* Cached balance, Life Cycle, Project, Dynamic Identifier, Key
- *Culture-dependent business objects:* Currency, Validation results
- *Commonly used financial business objects:* Bank accounts, Payment terms, Payment method, Settlement terms, Calendars (natural, fiscal, periodized), Financial batch, Financial Integration

The names of the Common Business Objects mostly speak for themselves.

2.3 Core Business Processes

The Core Business Processes layer provides business objects and default business logic for selected vertical domains. SanFrancisco delivers currently business components in the domains of accounts receivable, accounts payable, general ledger, order management (sales and purchase), and warehouse management.

Each domain process can be used independently or in conjunction with other processes. The currently implemented domains include:

– *Financials:* Ledger, Credit control, Item entry, Payment handling, Budgeting;
– *Warehouse Management:* Warehouse control, Stock replenishment, Picking Stock;
– *Order Management:* Quotations, Sales orders, Cash sales.

3 Architectural Aspects of SanFrancisco

Software architecture is the specification of the structure of a system. This can include: system organization, component structure, protocols for communication, data access, composition of design elements, etc. See [5]. We split the description of architectural aspects into two broad areas, *System organization* and *Component Structure.*

3.1 System Organization

The interesting aspects of the SanFrancisco system organization are: the Java environment on which it is built, its layered architecture, and its distribution infrastructure.

3.1.1 The Java Environment. The architecture of SanFrancisco is deeply rooted in the Java environment. By now, the advantages of Java are widely known. Java is a full-fledged Object Oriented language, accepted by the industry as the major application development language. It features portability, distributed objects, integration with databases, among others. The investment in the IT industry to build tools and applications for the Java environment is high.

3.1.2 Layered Architecture. The SanFrancisco components, which are developed as a pure Java framework, can exploit the advantages of the Java environment directly. We saw in Fig. 1, that the lowest layer of the SanFrancisco architecture consist of the Java virtual machine. The other layers of the SanFrancisco system architecture build on the Java layer, by providing successively: infrastructure functionality in the Foundation layer, common domain functionality in the Common Business Objects layer and specific domain functionality in the Business Process Layer.

3.1.3 Distribution. SanFrancisco has implemented a distribution infrastructure. The distribution facilities allow application developers to obtain Platform independence and Persistence for their application components. SanFrancisco provides the clients which make use of these application components with functions to create instances of distributed objects, refer to an existing persistent object, create new objects, and to invoke methods on distributed objects. Further, objects are uniquely identified in the network, and locking and commitment control or taken care of by SanFrancisco.

3.2 Component Structure

SanFrancisco is organized as an object-oriented Component Framework. The framework consists of a set of Business Components which interact with each other.

A component framework is not the same as a class library, which is familiar to most object developers. Class libraries are static structures and do not embody any kind of application flow. Business Process Components are dynamic, they make use of an architecture, the represent both design and implementation. Essentially, Business Process Components are like miniature applications, which can be customized and extended.

3.2.1 Model Driven Development. An important point to note here is, that the complete SanFrancisco framework is documented as an UML** model. New components to be used in application development will be added to the already existing SanFrancisco model during analysis. The implementation will use the model and implement the components as specified in the model using the rules in the SanFrancisco programming model.

What is the development process which is associated with the model driven development for SanFrancisco? A development process guides you through the process of building solutions on top of SanFrancisco business application components. It lists activities that you can do to follow and it indicates a logical sequence between some of these activities. The process used for SanFrancisco application development is not very different from standard object technology oriented development processes. A main point to note is, that it emphasizes *mapping* activities to SanFrancisco business components as part of the process.

If we restrict ourselves to the activities of application development itself (forgetting about assessments, project management, quality assurance, etc), we see the following activities:

- requirements collection,
- requirements mapping,
- analysis,
- design,
- coding and unit test.

After each of the activities, mapping to SanFrancisco components will occur. This can cause re-iteration through previous phases, to find the best fit to the framework.

3.2.2 Design Patterns. The SanFrancisco Framework makes extensive use of design patterns in order to provide an easy way to understand common object models and their respective solutions. Published design patterns are used [3] and new ones were created. For example, in the Foundation Layer, the *Factory* pattern is used to create business objects. The *Property Container* pattern allows applications to add attributes to an object at run-time of the application, without changing the basic structure of the object's class.

3.2.3 Framework Extensions. Application developers need a common way to extend Frameworks in order to preserve the design integrity of the original framework. In addition, common extension mechanisms provide:

- Consistency.
- Interoperability of applications from different vendors.
- Isolate framework changes made by application developers, to a limited number of classes. This makes maintenance easier and allows upward compatibility for new releases of the framework.

SanFrancisco provides extension points that identify where and how to extend the framework.

3.2.4 San Francisco Programming Models. One of the unique features of SanFrancisco is the ability to run in a network of interconnected servers, where the placement of data in containers is transparent to the application which accesses it. The San Francisco Programming Models supports consistent programming interfaces for using and developing business objects. This programming model prescribes both coding of the Business Components which reside on the server, and the programming conventions for a client which accesses these components.

3.2.5 SanFrancisco and Enterprise JavaBeans. The Enterprise JavaBeans** is a new component architecture [6] for scalable, transactional, and multi-user secure Java applications. The next version of SanFrancisco will be based on this important architecture. It will allow developers to use other EJB compliant components together with SanFrancisco components. Customers will have a choice of EJB-server which can be used to deploy SanFrancisco applications, instead of the proprietary SanFrancisco Foundation Layer.

4 Traditional Ways to Develop SanFrancisco Applications

Although the tradition is not very long, only a few years, it has become common practice in object-oriented application development to use an analysis/design tool such as Rational Rose or Select**, to generate code skeletons from the model specifications, and to use an IDE such as IBM's VisualAge for Java to complete the code and add a user interface.

With the SanFrancisco product a code generator is provided which generates 95 % of the server code required for a SanFrancisco application. Business logic has to be added to the code. In the latest release of SanFrancisco (v1.4), a bean wizard can generate SanFrancisco Beans to access the server objects, and Java Swing panels, which can be developed further, using VisualAge for Java. To produce well-performing, well-designed applications further coding and tuning has to be done by the developer.

When code skeletons are re-generated from the model, it is a problem to re-insert the business logic and other changes. They have to be done by a cut-and-paste process. A code generator which is able to preserve code is planned. For the bean wizard, there is no such plan yet.

4.1 A Wish List for SanFrancisco Development Tools

In the previous sections we described why application development, technology, and customers alike, are interested in using framework technology such as SanFrancisco provides. Despite the advantages a framework can offer, it has been found that in practice application development is more difficult than expected. There are two major reasons for this:

1. High learning curve for object technology and framework content
2. Tools are not well adapted to be used with frameworks yet.

Most major Modeling tool and IDE (Integrated Development Environment) vendors introduced new versions of their tools to accommodate the Java programming language and UML, the Universal Modeling Language. To adapt to large frameworks such as SanFrancisco, more is needed.

- Some modeling and development tools do not scale very towards 2000+ components (the number of components SanFrancisco provides).
- The same is true for code generation technology for such a large programming model as SanFrancisco defines. Many code generators provided by modeling tool vendors are not easily customizable. Therefore, it is often impossible or very difficult to adapt code generators to specific framework or user requirements.
- The gap between the modeling and programming tools is still wide. Many code generators require cutting and pasting after re-generation of code because of a design change.
- The tools are mostly not written in Java yet. They often run on a limited set of platforms, and have fixed functionality. They cannot integrate with the target environment.
- Support for components as black-box building blocks is lacking in most tools and in UML itself.

Because of these problems, good tools targeted at the SanFrancisco framework were slow to appear and the ones which are there now, still lack functionality.

It seems that there is room for a new tool architecture. This architecture should implement the following wish list:

- Able to run on all Java platforms.
- Full UML support.
- Full Java support.
- Supports components.
- Exploit new document facilities, and use them to store model information, program source code and design documentation. (think of XML and XMI to use here).
- Customizable to user requirements.
- Flexible towards technology changes..
- *One integrated,* but configurable tool, combining modeling and code development facilities.
- Generate code for both server and client code (for a variety of clients, both fat and thin).
- The code generator would be customizable by the developer.
- Changes would be preserved after code generation.
- The tool would be small, but scalable.
- The tool would integrate with SanFrancisco, allowing you to perform utility functions etc.

Do we need to create a monster to fulfill the requirements sketched here? We do not think so. An often overlooked feature of the Java language environment is that Java classes are linked dynamically into the runtime environment. No other third generation language has this feature, including C++ or Smalltalk. This allows a tool developer to develop a set of building block components for each target environment and package them according to the needs of a user. Massive recompilation, for which C++ is famous, or, the difficulty to accommodate conflicting requirements within one image, for which Smalltalk is known, are never a problem with Java.

Other opportunities in Java to make programs which are flexible and customizable, are its JavaBean facilities, which are the Java implementation of components, and resource bundles, which can store in a file translations for program items in various languages, or configuration directives.

The remainder of this article describes Business Component Prototyper. BC-Prototyper fulfills some of the requirements described above.

5 Business Component Prototyper

Business Component (BC-) Prototyper is an experiment in tool architecture, in an attempt to research some of the problems sketched above. It shows a proof of concept by providing an amazingly broad, but not very deep or polished, functionality. We do not pretend that BC-Prototyper has an answer for all items on the wish list, we just claim that our architecture would enable a solution to most requirements on the wish list. We also do not think BC-Prototyper can compete with any of the “big” tools, we just want to make clear that these tools could benefit from the ideas we provide.

Our proof that the BC-Prototyper architecture is effective, is mainly a hand-waving one. Showing its functionality, many people find it hard to believe that this tool was basically developed by just one person in spare time. Strict building block architecture, heavy use of code generation, and heavy use of the tool to build itself, have allowed to achieve these results.

Currently, Business Component Prototyper is an evaluation tool, provided with SanFrancisco v1.4. Its objective is to help new SanFrancisco developers to understand the SanFrancisco foundation layer programming model more easily, by experimenting with creating some prototypes, which actually can run. BC-Prototyper is intended for evaluation, education, and simple prototyping use. For the development of production applications, other modeling tools should be used. The version of BC-Prototyper described here has more functionality than the version provided with SanFrancisco. This functionality is experimental in nature.

5.1 Overview

It will be helpful to describe some of the functionality of BC-Prototyper first.

The BC-Prototyper is a pure Java tool that can run on any client platform supported by SanFrancisco. It is a simple modeling tool, featuring UML, and it has some integrated development environment (IDE) capabilities, such as editing or compiling Java code. Additionally, the tool has code generation facilities, and interfaces to many SanFrancisco utility functions. A major feature of BC-Prototyper is, that it can produce running applications, including a Graphical User Interface (GUI).

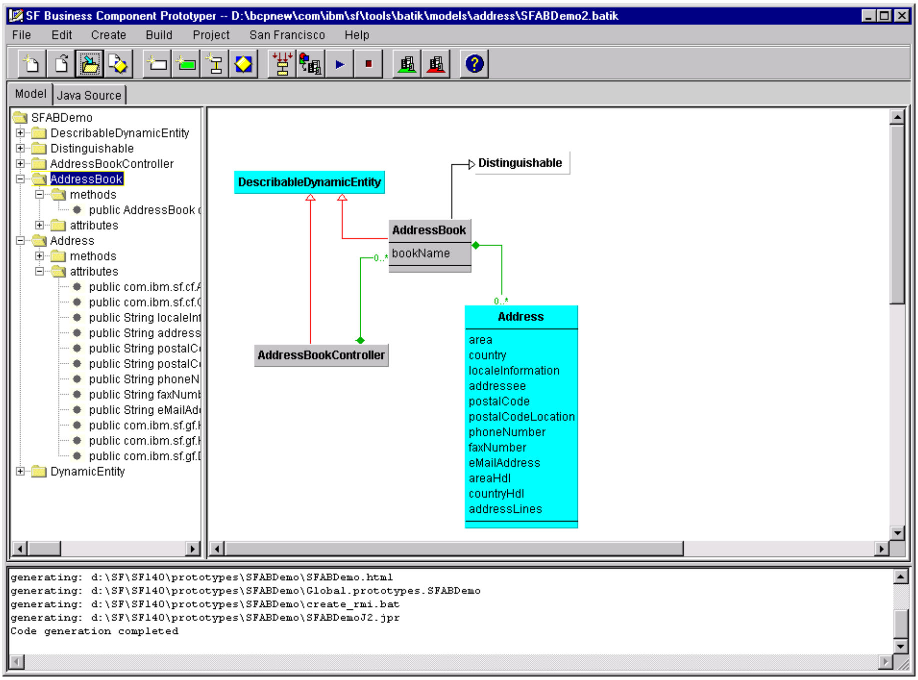
Very briefly, developing a SanFrancisco application using BC-Prototyper would look as follows:

- Develop a static object model.
- Generate and compile Java code.
- Generate proxies and add names of classes to the SanFrancisco naming service.
- Start and run the application.
- Iterate, to add business logic, or to customize the GUI.

5.2 BC-Prototyper Building Blocks

When the tool is started, a window will appear with three parts (see Fig. 2):

1. A menu bar and tool bar. The toolbar buttons show the main actions a user can perform in the SanFrancisco configuration of the tool, for example:
   - Create, open, or save a project, edit project properties.
   - Create new class or relationship, Import a component, edit any of these model items.
   - Perform a build for selected components (generate server, client and GUI code, compile, generate proxies).

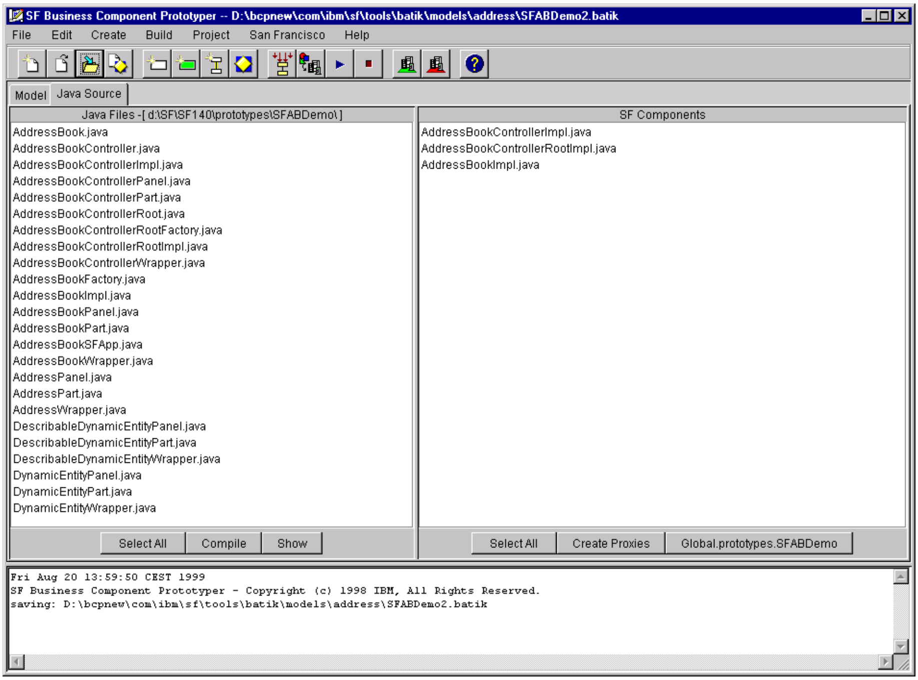


```
generating: d:\SF\SF140\prototypes\SFABDemo\SFABDemo.html
generating: d:\SF\SF140\prototypes\SFABDemo\Global.prototypes.SFABDemo
generating: d:\SF\SF140\prototypes\SFABDemo\create_rmi.bat
generating: d:\SF\SF140\prototypes\SFABDemo\SFABDemoJ2.jpr
Code generation completed
```

Fig. 2. BC-Prototyper window, the modeling tab-panel

   - SanFrancisco utilities: Start/Stop SanFrancisco, Append names to the SanFrancisco name space, Start/Stop the application, Show help information.
2. A workarea, with one or more tab-panels.
   - The *first* tab-panel has the modeling functionality. The first panel displays the currently loaded model in tree form and in UML-diagram form.
   - The *second* tab-panel provides Java development functions, and access to some of the SanFrancisco utility features. It shows a list of .java files in the package directory of the current project, a list of .java files for which proxies can be generated, buttons to compile selected .java files, show them in a simple editor, or generate a proxy. The generated file which contains the SanFrancisco naming information for the project, can also be viewed and updated in this panel.
3. A message area, where the actions the tool performs are logged, or where errors are displayed.

In Fig. 2, the modeling tab-panel is visible. It provides all usual modeling functions to create or edit a static UML model. In addition to what most modeling tools provide, such as creating a class, an attribute with scope, type etc., in BC-Prototyper you can specify how a class or attribute should be presented in the GUI.


SF Business Component Prototyper -- D:\bcpnew\com\ibm\sf\tools\batik\models\address\SFABDemo2.batik
File
Edit
Create
Build
Project
San Francisco
Help
Model
Java Source
Java Files -[d:\SF\SF140\prototypes\SFABDemo\ ]
SF Components
Select All
Compile
Show
Select All
Create Proxies
Global.prototypes.SFABDemo

```
AddressBook.java
AddressBookController.java
AddressBookControllerImpl.java
AddressBookControllerPanel.java
AddressBookControllerPart.java
AddressBookControllerRoot.java
AddressBookControllerRootFactory.java
AddressBookControllerRootImpl.java
AddressBookControllerWrapper.java
AddressBookFactory.java
AddressBookImpl.java
AddressBookPanel.java
AddressBookPart.java
AddressBookSFApp.java
AddressBookWrapper.java
AddressPanel.java
AddressPart.java
AddressWrapper.java
DescribableDynamicEntityPanel.java
DescribableDynamicEntityPart.java
DescribableDynamicEntityWrapper.java
DynamicEntityPanel.java
DynamicEntityPart.java
DynamicEntityWrapper.java
```

```
AddressBookControllerImpl.java
AddressBookControllerRootImpl.java
AddressBookImpl.java
```

```
Fri Aug 20 13:59:50 CEST 1999
SF Business Component Prototyper - Copyright (c) 1998 IBM, All Rights Reserved.
saving: D:\bcpnew\com\ibm\sf\tools\batik\models\address\SFABDemo2.batik
```

Fig. 3. The development tab-pane in BC-Prototyper

You can also specify specific SanFrancisco characteristics. For example, you can say that a class should be a SanFrancisco Entity or Dependent, or that a relation between two classes should be implemented as an EntityOwningExtent collection. For an attribute you can specify that in the GUI its value should be seen as a TextField or TextArea etc.

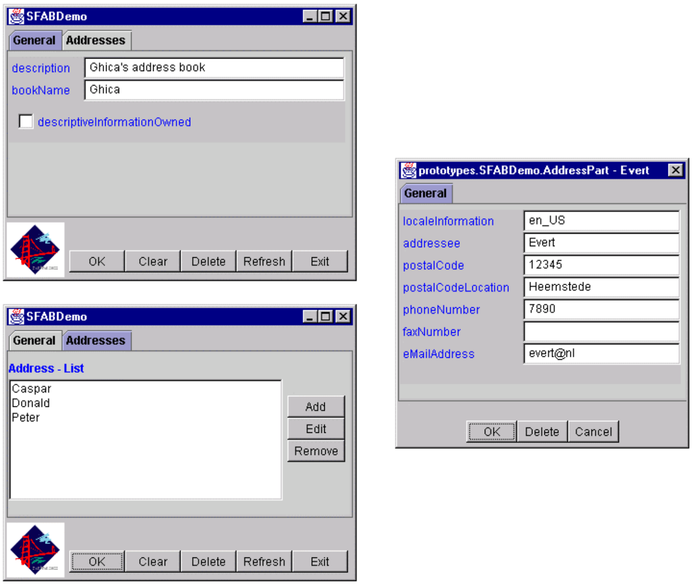
BC-Prototyper provides specific support for Components. The components are shown as green class shapes on the drawing area. In Fig. 2, Describable-DynamicEntity and Address are components, taken from the SanFrancisco framework. From a modeling point of view, components are just like classes which are predefined and put in a class library. Whether the class actually consist of one or 100 real classes is not important when you try to use one. What is important, is to know its interface and functionality. Unfortunately, the SanFrancisco components are not specified that way in the Rose** models provided with the product. You would see the complete static structure of a component when using Rational Rose. For BC-Prototyper, we have chosen to re-engineer the component specifications from the Java .class files. This has the added advantage that any .class file can be made into a component, not just SanFrancisco model fragments.

The white class shape shows an interface. Interfaces are predefined, like components. Additionally, interfaces can contain code generation fragments. This allows you to provide a default implementation for a Java interface. The developer usually does not have to write any code to implement the interface.

The tree view shows details of the model, updates can be made by double-clicking one of the items in the tree, which causes a property editor to appear. Note that the tree view shows a class, DynamicEntity, which is not present in the model diagram. This class is the superclass of DescribableDynamicEntity. It is needed to generate the correct code, but for readability reasons it is made invisible in the diagram.

When the specification of the UML model is complete, code can be generated. BC-Prototyper provides sets of code generation templates. The main sets available are templates for a stand-alone application such as BC-Prototyper itself, and to generate a SanFrancisco application. When SanFrancisco code generation is applied to the AddressBook example, a set of .java file will be generated, as shown in Fig. 3, and some other files, such as HTML documentation for the model.

When the code is generated and compiled, and proxies are created, the next thing to do, is to add the name tokens for the newly defined classes to the SanFrancisco naming configuration. A utility is provided to perform this function. The SanFrancisco servers will automatically be started if necessary.



SFABDemo

General | Addresses

description: Ghica's address book
bookName: Ghica
□ descriptiveInformationOwned

SFABDemo

General | Addresses

Address - List
Caspar
Donald
Peter

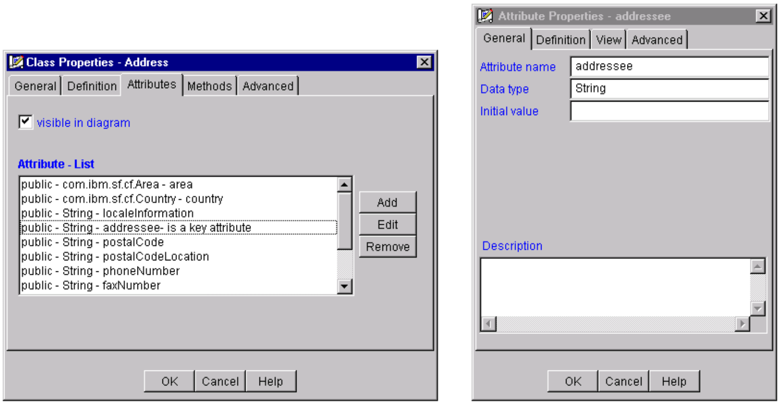
prototypes.SFABDemo.AddressPart - Evert

General

localeInformation: en_US
addressee: Evert
postalCode: 12345
postalCodeLocation: Heemstede
phoneNumber: 7890
faxNumber:
eMailAddress: evert@nl

Fig. 4. The generated GUI for the AddressBook Example

Next, the application can be started. The panels you may see in the running applications are the AddressBook panel and the Address panel. See Fig 4. Note that the fields in the Address panel are all generated from the information in the imported component.



☑ visible in diagram

Attribute - List

```
public - com.ibm.sf.cf.Area - area
public - com.ibm.sf.cf.Country - country
public - String - localeInformation
public - String - addressee- is a key attribute
public - String - postalCode
public - String - postalCodeLocation
public - String - phoneNumber
public - String - faxNumber
```

Attribute name	addressee
Data type	String
Initial value	

Description

Fig. 5. Generated GUI for BC-Prototyper itself

5.3 GUI Generation

BC-Prototyper generates at request of the user, not only model code which implements the structure of the model as defined by the user, but also a GUI, which can be used for later customization.

Simplified, the rules for generating a GUI are: A class in a model maps to a window, optionally with a set of tab-panels. Attributes in a class map to a TextField, TextArea, Checkbox, Choice, or Button, with a Label if appropriate. Classes with a contained one-many relationship, will map to a List in the containing class.

A simple example is the generated GUI for the AddressBook example. See Fig. 4. Note that we did not do anything to customize the generation. We could have entered nicer labels in our model, categorized the fields into more tab-panels, etc. The window for AddressBook contains fields which it inherits from DescribableDynamicEntity or DynamicEntity. If you click on the Addresses tab, you see the generated list for the one-to-many relationship to the Address component. From this panel you can open existing, or create new Address objects.

Another good example of a generated GUI, is the user interface of BC-Prototyper itself. When we apply the rules to the meta-model, we will obtain generated windows for the class property editor and attribute property editor as in Fig. 5. These windows are actually used as interface to BC-Prototyper to enter values into the model a user creates.

5.4 Use of Components in BC-Prototyper

Any development tool which supports a framework like IBM SanFrancisco, should have a strategy to support components.

The solution chosen for BC-Prototyper is to import information from .class files. This is a very general solution, it can be applied to any compiled Java file.

For the purpose of SanFrancisco, some of the programming model patterns have been applied in the re-engineering process. It was found that some information cannot be extracted fully from the class files (initialize(...) methods etc.). This information is added manually.

Once a component is put on the BC-Prototyper work area, they can be used as any class defined by the user. You should not change the contents for the server part, but you can change the view specification in the component.

In Fig. 2, you can see the list of Java source files for the Address component. An AddressPanel, AddressPart and AddressWrapper are generated, to provide client and GUI code for the component. Contrast this with AddressBook, a newly created Business Object, for which also AddressBook.java, AddressBookImpl.java and AddressBookFactory.java are generated.

6 The Architecture of BC-Prototyper

The architecture of BC-Prototyper can best be characterized as a *"spider"* architecture. The body of the spider consists of an object-oriented structure, which is an implementation of the meta-model of the tool. Its legs are filters through which models stored in some external format can be put into the body of the spider, or filters through which external representations can be generated.

The BC-Prototyper architecture is described in more detail in [2]. Here we give an overview of the main items.

The design of BC-Prototyper shows four major parts:

1. A mini-framework for use by BC-Prototyper itself, and by the applications which are built with it.
2. The spider body, consisting of generated code from the meta-model. It implements the data structures for the model information, and the user interface in the form of property editors for the model.
3. Core tool functionality, such as the code generator- template interpreter and model drawing routines.
4. A set of configurable, bean-like modules, the plugin adapters, encapsulating menu options and/or toolbar button actions, or tab-panel functionality. An .ini file determines at start-up what the functionality of BC-Prototyper will be, by providing a list of plugins to include.

The external file formats can include Java source files, Java class files, HTML documents, XML documents, .bat files and others. The spider legs are more technically called plugin-adapters. Plugins are coded as JavaBeans. The availability of a plugin is determined by a configuration file, which is read when the tool starts. In this way, the tool can be configured for different purposes or for different code generation target environments.

6.1 The Code Generator

We described the GUI generation rules and the use of components in BC-Prototyper as functional items in the previous section. We consider code generation an architectural feature of BC-Prototyper, because it had a profound influence on the way the tool was developed.

The server-side programming model of SanFrancisco is not very complex, but large in size. Client code for transaction management and GUI adds considerably to the complexity of the code. For our experiment in architecture, a very efficient approach would be necessary, and we found it in using XML code generation templates.

Code generation with XML templates in BC-Prototyper is an interaction between the contents of these templates, a template interpreter (written in Java) and introspection into the meta-model for in the tool. The template contents are Java source code (or HTML, or project files for an IDE, or SanFrancisco configuration files or whatever you need), surrounded with XML tags, and in some places the source code is replaced by XML-elements.

To get a flavor of what these templates are like, here is a snippet of a template for the simplest of examples. It generates "getter" methods for all public attributes for a class in the model:

```
<Rule>
<Target>Attributes</Target>
<Condition> scope = public </Condition>
public &type; get&u.name;() {
return  iv&u.name;;
}
</Rule>
```

The u. is a shorthand for putting the first character of the replacement value in uppercase. Further, the snippet should be self-explanatory. For example, if the name of one of the public attributes for an Address class was "city", and its type "String", then this template would generate:

```
public  String getCity() {
    return ivCity;
}
```

The effect the adoption of this code generation technique had on Business Component Prototyper was large. Not only was the author able to complete a code generator for SanFrancisco in a very short period, with more functionality than any other tool could provide at the time. The author was also able to develop BC-Prototyper itself into a professional looking tool in the same short period. Why? Because changes to the meta-model of the tool, and to the code generator, are immediately available to the tool itself. And the tool is developed using the tool. This had a snow-ball effect on the tool development.

The technical reason that XML in conjunction with introspection in Java is very effective, may be the following:

- Generated source code or documentation is a flat, serialized representation of an object structure. The mapping of the object structure to a flat representation can be fairly complex. An XML template is an implicit implementation of this mapping. When the mapping is already done, the generator can be simple.
- XML templates are readable and editable by humans. By changing the XML templates, you can change the mapping, and therefore the generated code, without having to re-code or recompile the code generator. New generators are easily made by adding tags to existing source code or documentation. Users can specify their own code generation, or customize templates provided easily.

Part of the effectiveness of the tool development is also its “spider” architecture. The body of the spider contains the generated code from the meta-model, and is easily adaptable by adapting the meta-model or the code generation template. Any number of legs can attach to the spider body through a simple bean interface. The legs can contain filters to import/export XML documents, Java code etc., to/from the spider body.

6.2 A Look Back at the Wish List

When we match the functionality and architecture of BC-Prototyper to our wish list, described in Section 4.1, we find that many of our wishes are indeed fulfilled:

- BC-Prototyper runs on all Java platforms because it is written in Java
- It certainly does *not* provide full UML support. Its extensible architecture would allow for addition of Object Interaction Diagrams, State Diagrams, or any other UML feature quite easily. To develop BC-Prototyper into a full tool instead of a prototype, these functions must be added. The code generation for SanFrancisco application development totally depends on a static object model, which is provided with BC-Prototyper, as shown in Fig. 2.
- Java development support is primitive. The same argument applies here as for the previous point.
- BC-Prototyper does support components. And it does so in a more transparent way than other tools the author is aware of.
- XML is used both for storing the model itself, and for encoding code generation templates.
- A knowledgeable user could adapt the meta-model or the generation templates. In this way the tool can be customized and extended.
- Technology changes can be cared for in the same way as the previous point, as long as Java is in the center of it.
- BC-Prototyper is one integrated tool, but configurable, by deciding which plugin components will be included at startup.

- We did not mention it in the functional description, but changes can be preserved by the tool. Generated code can be imported into any other IDE, changed, and imported back into BC-Prototyper. Code changes will be put back into the model at the right spot, provided the user has indicated changes with a change tag.
- Integration with SanFrancisco is provided for as described in Section 5.

7 Conclusion

This article described SanFrancisco, a new large, Java based component framework. We looked at the architecture of SanFrancisco, and derived from it the features an effective development tool for Business Component frameworks should have.

We found that, although not complete yet, Business Component Prototyper can provide the necessary features to become a good development tool for Java framework applications in general and for SanFrancisco in particular. BC-Prototyper has functionality which is usually not found in other development tools, such as generation of complete, running prototypes, including GUI, and tight integration with SanFrancisco.

Trademark Summary

References

1. Grady Booch, Ivar Jacobson, James Rumbaugh. *The Unified Modeling Language User Guide.* Addison-Wesley (1998).
2. Ghica van Emde Boas. *SF Business Component Prototyper, An Experiment in Architecture for Application Development Tools.* Submitted to: IBM Systems Journal (2000). 145
3. Gamma, Helm, et al. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley (1994). 137
4. IBM SanFrancisco: http://www.software.ibm.com/ad/SanFrancisco. 133
5. Mary Shaw, David Garlan. *Software Architecture, Perspectives on an Emerging Discipline.* Prentice Hall (1995). 135
6. Sun Microsystems, Enterprise JavaBeans Specification, version 1.1 public draft (1999). 137

7. W3C, Extensible Markup Language (XML) 1.0.
http://www.w3.org/TR/1998/REC-xml-1998210.
8. XMI, XML Model Interchange.
http://www.software.ibm.com/ad/features/xmi.html

Databases and the World Wide Web

Paolo Atzeni

Dipartimento di Informatica e Automazione
Università di Roma Tre
Via della Vasca Navale, 79
00146 Roma, Italy
atzeni@dia.uniroma3.it
www.dia.uniroma3.it/~atzeni/

Abstract. The World Wide Web is likely to become the standard platform for future generation applications. Specifically, it will be the uniform interface for sharing data in networks, both Internet and intranet. From a database point of view, there are two main directions of interest: the first one is related to the extraction of information from the Web (possibly in order to manage it by means of database techniques and tools) and second to the database support (again in terms of both methods and tools) to the management of data to be offered through the Web. A major, preliminary issue is the method used for the description of data, where the discussion between structured and semistructured approaches. Interesting developments will probably by stimulated by the advent of XML.

1 Introduction

It is now generally agreed that database research produced very interesting results, which provided the basis for a very solid technology (interesting reports on the issue have appeared in the last ten years: Silberschatz et al. [39,40] and Bernstein et al. [12]. Databases were conceived and developed back in the 1960s and 1970s in order to support the integrated management of data in business applications, such as banking, reservations, personnel, inventory management. In a few words, we can say that the goal of database technology is to produce systems for the management of large volumes of shared and persistent data with reliability (ensuring that data is not lost) and power (in terms of both efficiency, say, throughput and response time, and effectiveness, say, usability and flexibility). Major achievements in this were respect were obtained in the 1980s, with the development and acceptance of relational systems, which offer high-level languages and an associated technology for reliable and efficient transaction management, both in centralized and distributed frameworks. Properties of databases are now duscussed at length in textbooks, such as Atzeni et al. [3] or ElMasri and Navathe [19].

In the last decade, it has been recognized that the scope of database technology is much larger than the basic business world that supported its initial

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 150–162, 1999.

development (see the aforementioned reports [39,40] for extensive discussions). The continuous reduction in the price/performance ratio for all hardware components has made it possible to apply database ideas to many applications, with more complex data, such as text, images, video, and scientific data sets. Then, the explosion of the Internet and of the Web has produced an even bigger spur to database technology: the Web offers the potential for integrating data from an enormous number of sources and for connecting huge numbers of clients (not only human users, but small powerful tools embedded everywhere, say in cellular phones, cars, domestic appliances). In this framework, the recent so-called "Asilomar report" (Bernstein et al. [12]) recommends the following as a ten-year goal for the database research community: "The Information Utility: Make it easy for everyone to store, organize, access, and analyze the majority of human information online." With the assumption that the majority of human information will be on the Web within a decade, this would mean that it will be through the Web (not necessarily today's Web, but what the Web will be by that time) that all this should happen. In synthesis, we could say that information systems will be "Web-based," in the sense that the Web will one of their interfaces (possibly the major or even the only one). In fact, the notion of a "Web information system" has been proposed, mainly to stress the original features and requirements that are emerging (Isakowitz et al. [28] present a collection of articles on the topic).

Bernstein et al. [12] mention a number of general themes of interest for future research in the database field:

- development of "plug and play" database systems, aimed at simplifying the management and administration and at supporting the understanding of the available information, and the integrability of autonomous sources;
- scalability of the approaches to federated database systems (conceived for the cooperation of a few systems, but now potentially involving thousands or even millions); this refers to both performance issues and semantic aspects, for example in the management of heterogeneous or irregular information;
- general revision of the overall architecture of database systems, because of the evolution of hardware and of the growth of the size of databases;
- better integration of data and processes, both with respect to conventional applications and to newer approaches, such as workflows and business rules;
- integration of structured and semistructured data, mainly as a consequence of the advent of XML.

Most of these themes include aspects that are directly related to the specific role databases have in WIS.

In this paper, we will try to point at some of the activities in the research field that tries to consider database methods and techniques applied to WIS, as well some of the lines currently being pursued. A word of caution is needed: given the high dynamicity of the field, we do not aim at a complete survey—we will mainly discuss the issues on the basis of the experience we had at Università Roma Tre within the Araneus Project (Atzeni et al. [6,7], Mecca et al. [33,35]). For other references on the subject, we mention a survey by Florescu et al. [22]

and the proceedings of two recent workshops (Atzeni et al. [9] and Cluet and Milo [15]).

We will begin by discussing in Section 2 the idea of *Web information systems*. Then, the major research issues will be described in Section 3, and specifically: in Section 3.1 we will discuss the role models play in this field, in Section 3.2 we will consider the problem of extracting data from the Web and in Section 3.3 the converse problem of designing and maintaining a Web site. We will conclude in Section 4 by briefly mentioning the impact XML might have on the field.

2 Web Information Systems

As we said in the introduction, it is probably the case that in a few years all information systems will be Web-based. At that point, the notion of Web information system could become so pervasive to be irrelevant (in the sense that no system would be outside this category). However, we believe that it deserves our attention now, for at least two reasons. First, the evolution of Web-based systems from simple, hand-produced sites to complex information systems has been gradual, and the various stages show some individual interest, in terms of requirements, methods and tools. This is especially true with respect to data management issues. Second, the Web has offered opportunities for new systems and services that would not have been possible in other environments.

In the same way as information systems are sometimes classified on the basis of their features, we believe it can be useful to classify WIS according to the complexity of the involved data and processes as shown in Figure 1 (Mecca et al. [35]). We discuss this classification by referring also to the way Web sites have evolved in the few years that have passed since its invention (see also Atzeni et al. [3, Section 14.2]).

Indeed, the first Web sites were composed of manually-crafted hypertexts, prepared with the goal of making information available. Pages were generally prepared in an ad-hoc way, possibly using information already available (although often to be reorganized). They would usually fit within the lower-left category (*Web-presence sites*, with low complexity of both data and processes). Many sites with these features still exist, mainly with marketing or basic advertising goals. In some cases, they are used as “entrance gates” to other systems.

Following the initial goal of the Web, the complexity of the data to be disseminated grew, and many *catalogue sites* (upper left part of the diagram) appeared. In this framework, it was soon realized that it could be useful to store in databases the information to be “published” on the Web, for a number of reasons: on the one hand because the database could already exist (or at least it could be created with multiple objectives, one being publication on the Web); on the other hand, a database could support the management of changes in data (especially if the structure and appearance of the hypertext stay unchanged) as well as a non-redundant support for redundant data (the Web can be seen as a “hypertextual view” over the database, and it is often the case that redun-

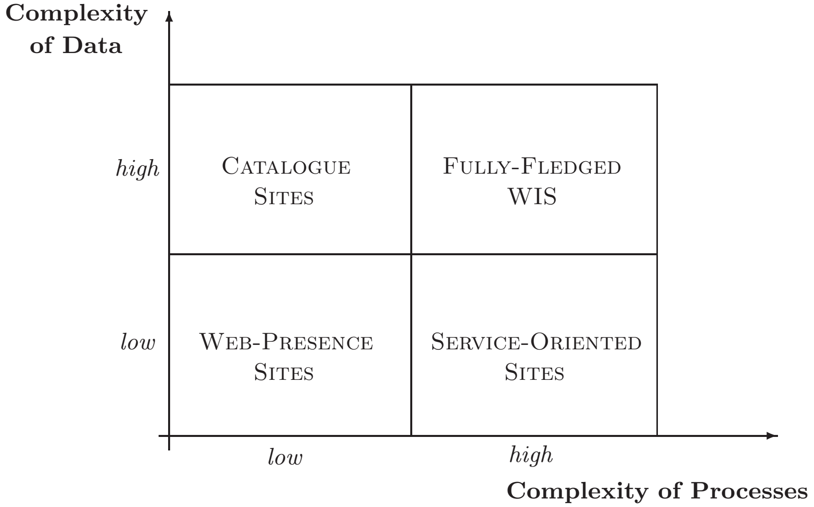


Fig. 1. Classification of Web Information Systems

dancies appear in the Web, at least to provide support for navigation, whereas databases tend to avoid them).

At the same time, the Web spurred the development of many services, along the lines already being pursued in the Internet world: the Web offered a much more friendly interface, and therefore there was an immediate explosion of initiatives. Services of various types appeared: from spontaneous discussion groups to search engines to free email providers. Here (despite the fact that there is often a database in the background) the visible complexity is only in the processes: we are in the lower-right portion of Figure 1.

The route towards fully-fledged WIS (upper-right part) is still to be completed, and database research has considered only in a marginal way the complexity of processes and applications. Indeed, attention has been mainly on features that extend, to the Web world, database services that are popular in traditional information systems. However, some steps have been made. A significant category is represented by many catalogue sites that allow for updates and transactions, at least from two points of view. On the one hand, modifications to the catalogue are often allowed by means of pages similar to those used for publication. This is not a minor point as it might appear at first sight, because content maintenance is a common problem in many Web sites (especially from the organizational point of view) and also because this was a first step toward the idea that most (if not all) interfaces could be Web-based. Usually, this feature remains within the domain of the organization that owns the site, but can be physically distributed, without any difficulty. A second, more glamorous, direction of extension for catalogues has been the growth of "electronic commerce," with a tremendous impact on the popularity of the Web, providing an indication of how the Web could change the way many activities are conducted (an interesting collection of articles has recently been edited by Dogac [18]). Indeed, there are two major forms of electronic commerce: the most popular among individual Internet users is probably that related to catalogue sales (which could be seen as an evolution of traditional mail order services), often referred to as *business to customer (B2C)*, whereas the most important with respect to volumes of exchanges and amount of money is that involving organizations at both ends, *business to business (B2B)*.

More general issues related to WIS are still emerging, and we believe that, for the time being, it would be difficult to indicate general topics, except for a comprehensive need for the integrated management of data and processes. This is indeed a recurring theme in database research, as it was one of the motivations for the development of object-oriented databases (see for example Bancilhon [10]) and of active-rule components for databases (Widom and Ceri [42]). As we mentioned in the introduction, the Asilomar report [12, Section 3.4] also emphasizes this requirement, indicating topics such as workflow support (of business rules), component models (such as CORBA, OLE, Jini), visual programming methodologies, and persistent programming languages.

A major aspect that needs to be mentioned is the fact that the Web (and more generally the Internet) provides access to many *autonomous* systems, which could interact with one another. Indeed, the idea of "cooperative information systems" has been proposed with reference to existing systems that cooperate (see De Michelis et al. [17]); well, in the Web-world we can say that, in principle, all systems cooperate. Cooperation can be loose (as extreme, lists of links to autonomous sites or services) or tight (integrated interfaces to heterogeneous systems, with differences made transparent). For example, the B2B form of electronic commerce requires a form of cooperation between the information systems of the involved enterprises. Similarly, most on-line catalogues (in a B2C system) are indeed gateways towards multiple autonomous systems.

3 Databases in WIS

A feature that is common to most of the applications we mentioned in the previous section is the need for an integrated management of data of various nature, from traditional "database-data" to data embedded in documents (for example HTML ones). We have introduced the term *Web-bases* (Mecca et al. [33]), as collections of data of heterogeneous nature, with various degrees of *semistructuredness* (see Suciu [41] for references and discussion), from highly structured to unstructured. More generally, the Web involves many information sources, which could be more or less structured, and the major need is to let these sources cooperate. In the process of organizing cooperation, if we mainly refer to the complexity of data, a crucial issue that arise is the exchange of data, and its transformation, for example from a structured form (say, a database) to an unstructured one (Web pages) or viceversa. In other words, an interesting goal could be the following: "treat database data as if it were Web hypertexts, and viceversa." Indeed, databases and hypertexts have each its own advantages and disadvantages: in some cases navigation over hypertexts is what users need (and may be data is in non-browsable databases), in other cases users need to correlate, sort and aggregate (in a word, "query") data (and they have them embedded in documents). Therefore, a fundamental feature that is needed for the management of Web-bases is the support to translations: from Web sites to databases and viceversa. We have already argued for the need of the former when we said that most Web sites (catalogues, for example) are supported by databases; as regards the latter, let us just mention the issue of *Web farming* (discussed at length by Hackatorn [25]): this is the process of feeding data warehouses, that is, special databases used for decision support (see Inmon [27]).

Therefore, we will devote our attention to the two basic steps:

- from Web-hypertexts to databases: querying and extracting data from the Web;
- from databases to hypertexts: generating and managing Web sites.

With respect to Web sites, the two steps could be described as bottom-up (extracting data) and top-down (producing HTML pages), respectively.

A third major aspect often cited (Mecca et al. [33], Florescu et al. [22]) in this framework is that related to integration. We have already mentioned its importance; however, we believe that here the challenges specifically related to the Web are mainly due to the transformations from one form to another, rather than to integration itself (which is a complex problem indeed, regardless of whether the sources are on the Web or not). Therefore, for the sake of brevity, we will not comment more on this issue; the interested reader can consult the above sources, especially the survey by Florescu et al. [22], for more information.

3.1 Data Modeling in WIS

A general point to be discussed here is the role abstraction plays in this framework; abstraction is a major feature in the database field, as it forms the basis for modeling and therefore describing the common features of similar database items. As we mentioned in the previous section, data in the Web is often unstructured or semistructured, in the sense that it shows little or no regularity. As a consequence, the Web has sometimes been described by means of a very simple data model, graph based (Mendelzon et al. [36]): Web pages are the nodes of the graph and hypertextual links are its edges; all internal structure (if any) is ignored. This is clearly possible in all cases, but does not help much in the organization of the process of extraction of data from Web sources. In fact, it is often the case that Web sites do show some degree of structure. At the opposite extreme, one can see that some Web sites show (at least in some of their pages, or in portions of them) well structured pieces of information (for example organized as tables). There are variuos reasons for having more structured sites: from the owner point of view, this may just be the consequence that the site is generated automatically by means of programs, and therefore the organization of pages is regular; the motivation can also be deeper, and in the user's interest: regularity in the structure helps in the navigation process. With this in mind, in the Araneus project (Atzeni et al. [6]) we found that it would be useful to have a model for the description of Web sites. Initially, this was conceived as a basis for querying: in fact the idea is that structure (and data models) can be used in as a documentation of a site (to support the user's understanding of the content), as a basis for querying (from the user point of view) and as a basis for query optimization (from the system point of view). Soon after that, we realized that modeling could be useful also in the process of generating a site with some regularity in the structure (as it is often needed in catalogue sites, as we said in Section 2). While referring the reader to our papers (Atzeni et al. [6], Mecca et al. [33]) for examples, we comment here on the major features of our model, called the *Araneus data model* (ADM). It is a "complex-object data model," in the spirit of ODMG or SQL3. Its major construct is the *page scheme*, whose instances, the pages, are essentially complex objects, with the as identifier and simple or complex attributes. Simple attributes can be ordinary values (numbers, text, images, etc.) or links, which are loosely typed (in the sense that the destination page can belong to one of a set of given page schemes). Complex attributes are tuples or lists. The model also provides constructs for modelling Web-specific features, such as maps and forms.

Other authors have advocated the adoption of *semistructured* data models: this is an idea that was originally conceived as the basis for the integration of heterogeneous data sources (Papakonstantinou et al. [37]) and makes use of a representation of data by means of labeled directed graphs (see Abiteboul [1] and Buneman [13]), where labels represent "attribute names," in database terminology, and there is no restriction on the labels; therefore, at least in principle, there is no schema on the data. An advantage of this approach is that all sets of data, no matter how irregular, can be easily represented. The main disadvantage is that there is no way to exploit regularity, when there is some. We believe that there is no definite answer on whether a structured or a semistructured approach is to be preferred, in general. The major issue is to be able to take advantage of regularity and structure whenever possible.

As we will discuss in the next subsections, we propose the use of a data model both in the top-down and in the bottom-up step. It is important to note that, in the latter case, the model can be used an abstraction tool, in the sense that not all aspects need be represented: this is obvious if we refer to graphical or presentation features, but can also be important if there are details (possibly unstructured) that turn out to be marginal (or may be difficult to model): they can simply be ignored.

3.2 From the Web to Databases: Extraction and Querying

Various approaches have been proposed to query the Web and to extract information from its sites. The first proposals (Konopnicki and Shmueli [30], Mendelzon et al. [36], Lakshmanan et al. [31]) considered the Web according to the graph model, as we discussed in the previous section, and therefore allowed conditions on paths on the graph as well as simple operations (or calls to external programs) within pages. The major limitation of these proposals was the difficulty to take into account the internal structure of pages. Also, the results of queries were essentially lists of pages. Subsequent proposals (including WebOQL, by Arocena et al. [2], StruQL, by Fernandez et al. [20], and our proposal Ulixes, Atzeni et al. [4,6]) aimed at taking the internal structure of pages into account and provide a structure also for their results.

The crucial point here is not related to deciding which could be the best model or the best language, but to find means for a suitable mapping from Web pages to the language itself: the point is that we need to transform unstructured (or at best, loosely structured) pages in order to be able to extract from them the pieces of data that are embedded. This is the task of *wrappers* (Hammer et al. [26]). In our approach, querying is based on ADM, which is a logical model, and wrappers are needed in order to map logical accesses to attribute values in ADM objects to physical accesses to HTML text. With respect to wrappers, after a first proposal based on a procedural language inspired by editor based operations, such as "cut and paste" (Atzeni and Mecca [5,32]), we resorted to a grammar-based formalism with exceptions (Crescenzi and Mecca [16]): it allows a declarative description of pages (using a context-free grammar), but can handle irregularities by means of an exception handling mechanism. This has turned out to be a reasonable compromise in the description of sites.

At this point, the query process can be formulated by means of a query language whose syntax resembles known SQL-like syntax. The result, in our current prototype, is a relational table, but extensions to a complex-object model could be easily produced, but they would not add much to the major contribution of the approach, which is in the possibility of extracting "database data" from hypertexts, and therefore the major feature is wrapper technology.

3.3 From Databases to the Web: Generating and Managing Sites

With resepct to the "top-down" problem, generating Web sites from database, it is important to say that there are now many tools available on the market. However, we do believe that tools mainly provide very low-level support (in terms of HTML development) or, at best, simple mappings that allow to build Web interfaces to database data, but with no systematic support, especially with respect to the overall organization of the site.

It turns out that most Web sites do not satisfy user needs: the information kept is poorly organized and difficult to access; also it is often out-of-date, because of obsolete content and broken links. In general, this is a consequence of difficulties in the management of the site, both in terms of maintaining the structure and of updating the information. Many Web sites exist that have essentially been abandoned.

We believe that this situation is caused by the absence of a sound methodological foundation, as opposed to what is now standard for traditional information systems. In fact, Web sites are complex systems, and, in the same way as every other complex system, they need to be developed in a disciplined way, organized in phases, each devoted to a specific aspect of the system. In a Web site, there are at least three components (see Atzeni et al. [8], Fernandez et al. [21], Ceri et al. [14]): the information to be published (possibly kept in a database); the hypertextual structure (describing pages and access paths); the presentation (the graphical layout of pages). These issues have led us to develop a methodology (Atzeni et al. [7]) that is based on a clear separation among three well distinguished and yet tightly interconnected design tasks: the database design, the hypertext design, and the presentation design. Now, it is widely accepted that data are described by means of models and schemes, at various levels, for example conceptual, with the ER model and logical level, with the relational model (see Batini et al. [11] for a description of design methods). In a data-intensive Web site, it is common to have large sets of pages that contain data with the same structure (coming from tuples of the same relation, if there is a database): therefore, as we have already argued, we believe that models are useful description of hypertexts. We have already briefly illustrated ADM, to be considered a logical model for hypertext (the counterpart, in this framework, of the relational model for data). However, given the various facets that can arise, we believe that hypertexts also need, in the same way as data, to be described at a higher, conceptual level. Therefore, our methodology makes use of a conceptual model for hypertexts (called NCM, the *Navigation Conceptual Model*), which is essentially a variation of the ER model suitable to describe hypertextual features in an implementation independent way. In summary, the various phases of the methodology are shown in Figure 2 with the precedences among them and their major products (schemes according to appropriate models).

It is worth noting that other proposals have been recently published that present some similarities, though with a less detailed articulation of models (Fraternali and Paolini [23], Fernandez et al. [21]). The origins of the methodological aspects can be traced back to previous work on hypermedia design (Garzotto et al. [24], Isakowitz et al. [29], Schwabe and Rossi [38]). Most of these approaches, including ours, also provide languages, or at least tools, for the generation of Web sites from the declarative description of its structure, by supporting the mapping from the database to the site. This direction of mapping is clearly easier than that discussed in the previous section with respect to wrapping, because everything is under control. The tools also support the separation between structure and presentation, by providing some forms of *template pages* that describe the common appearence of pages with the same structure (in our approach, all the pages with the same page schema [9]).

4 The Next Step: XML

We conlude by briefly noting that the recent introduction of XML, which could emerge as a new standard for data representation and exchange, is stimulating a lot of interest in the database research community. Two major features are relevant here: first, XML offers (by means of DTDs) the possibility of describ-

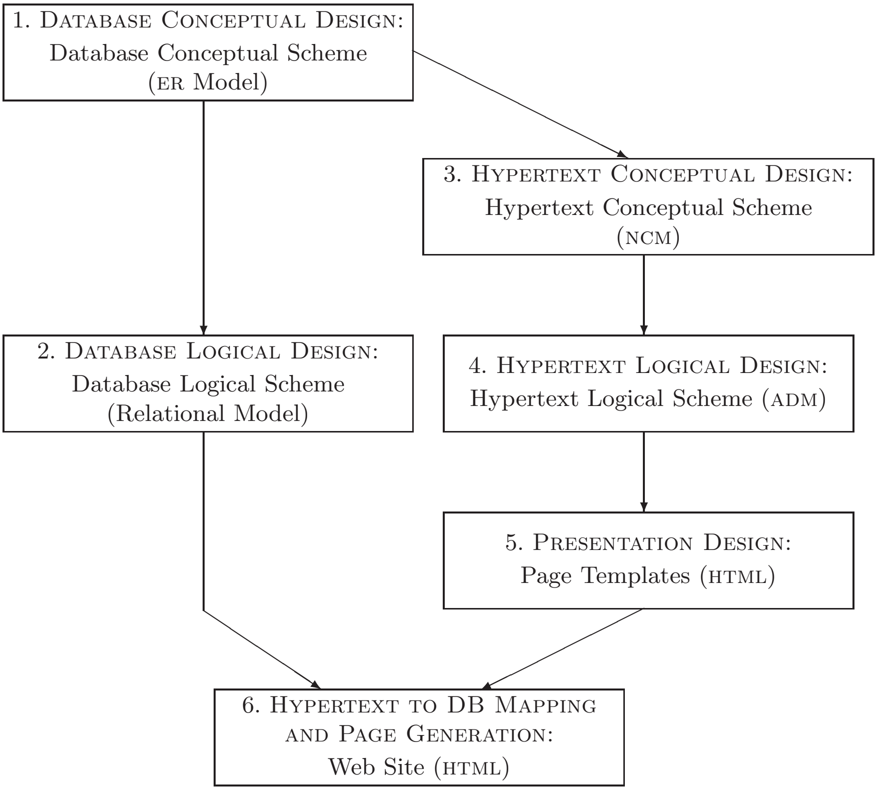

1. Database Conceptual Design:
Database Conceptual Scheme
(er Model)
3. Hypertext Conceptual Design:
Hypertext Conceptual Scheme
(ncm)
2. Database Logical Design:
Database Logical Scheme
(Relational Model)
4. Hypertext Logical Design:
Hypertext Logical Scheme (adm)
5. Presentation Design:
Page Templates (html)
6. Hypertext to DB Mapping
and Page Generation:
Web Site (html)

Fig. 2. The Araneus Design Methodology

ing the content of Web pages more accurately than HTML; second, XML provides a separation between content and presentation (for example by means of XSL, eXtensible Stylesheet Language). As we have recently discussed (Mecca et al. [34]), both these features can be exploited in an interesting way as extensions of current approaches. In particular, DTDs and descriptions of sources can be very useful in supporting the automation of the wrapping process. However, a major problem would remain, the need for understanding structure and using the structure provided by by an external, autonomous source. Also, the difficulty of choosing between a structured and a semistructured approach would not be reduced.

Acknowledgments

I would like to thank Gianni Mecca and Paolo Merialdo, together with whom I carried out my research activity in this area.

References

1. S. Abiteboul. Querying semistructured data. In *Sixth International Conference on Data Base Theory, (ICDT '97), Delphi (Greece), Lecture Notes in Computer Science, Springer-Verlag*, 1997. 156
2. G. O. Arocena and A. O. Mendelzon. WebOQL: Restructuring documents, databases and Webs. In *Fourteenth IEEE International Conference on Data Engineering (ICDE'98), Orlando, Florida*, 1998. 157
3. P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone. *Database Systems: Concepts, Languages and Architectures*. McGraw-Hill, London, 1999. 150, 152
4. P. Atzeni, A. Masci, G. Mecca, P. Merialdo, and E. Tabet. Ulixes: Building relational views over the Web. In *Thirteenth IEEE International Conference on Data Engineering (ICDE'97), Birmingham, UK*, 1997. Exhibition Program. 157
5. P. Atzeni and G. Mecca. Cut and Paste. In *Sixteenth ACM SIGMOD Intern. Symposium on Principles of Database Systems (PODS'97), Tucson, Arizona*, pages 144–153, 1997. 157
6. P. Atzeni, G. Mecca, and P. Merialdo. To Weave the Web. In *International Conf. on Very Large Data Bases (VLDB'97), Athens, Greece, August 26–29*, pages 206–215, 1997. 151, 156, 156, 157
7. P. Atzeni, G. Mecca, and P. Merialdo. Design and maintenance of data-intensive Web sites. In *Int. Conf. on Extending Database Technology (EDBT'98), Valencia, Lecture Notes in Computer Science 1377*. Springer-Verlag, 1998. 151, 158
8. P. Atzeni, G. Mecca, and P. Merialdo. Design and maintenance of data-intensive Web sites. In *VI Intl. Conference on Extending Database Technology (EDBT'98), Valencia, Spain, March 23–27*, 1998. 158
9. P. Atzeni, A.O. Mendelzon, and G. Mecca, editors. *The World Wide Web and Databases, International Workshop WebDB'98, Valencia, Spain, March 27–28, 1998, Selected Papers. Lecture Notes in Computer Science 1590*. Springer-Verlag, 1999. 152, 158
10. F. Bancilhon. Object-oriented database systems. In *Seventh ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'88)*, pages 152–162, 1988. 154
11. C. Batini, S. Ceri, and S.B. Navathe. *Database Design with the Entity-Relationship Model*. Benjamin and Cummings Publ. Co., Menlo Park, California, 1992. 158
12. P. Bernstein et al. The Asilomar report on database research. *ACM SIGMOD Record*, 27(4):74–80, December 1998. 150, 151, 151, 154
13. P. Buneman. Semistructured data. In *Sixteenth ACM SIGMOD Intern. Symposium on Principles of Database Systems (PODS'97), Tucson, Arizona*, pages 117–121, 1997. 156
14. S. Ceri, P. Fraternali, and S. Paraboschi. Design principles for data-intensive Web sites. *ACM SIGMOD Record*, 28(1):84–89, March 1999. 158
15. S. Cluet and T. Milo, editors. *ACM SIGMOD Workshop on The Web and Databases (WebDB'99), June 3–4, 1999, Philadelphia, Pennsylvania, USA, Informal Proceedings*. INRIA, 1999. 152
16. V. Crescenzi and G. Mecca. Grammars have exceptions. *Information Systems*, 23(8):539–565, 1998. Special Issue on Semistructured Data. 157
17. G. De Michelis, E. Dubois, M. Jarke, F. Matthes, J. Mylopoulos, M. Papazoglou, K. Pohl, J.-W. Schmidt, C. Woo, and E. Yu. Cooperative information systems: A manifesto. In M. Papazoglu and G. Schlageter, editors, *Cooperative Information Systems: Trends & Directions*, pages 315–363. Academic Press, New York, 1998. 154

18. A. Dogac. Special section on electronic commerce, introduction. *ACM SIGMOD Record*, 27(4):5–6, December 1998. 154

19. R.A. ElMasri and S.B. Navathe. *Fundamentals of Database Systems*. Benjamin and Cummings Publ. Co., Menlo Park, California, second edition, 1994. 150

20. M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for a Web-site management system. *ACM SIGMOD Record*, 26(3):4–11, September 1997. 157

21. M. F. Fernandez, D. Florescu, J. Kang, A. Y. Levy, and D. Suciu. Catching the boat with Strudel: Experiences with a Web-site management system. In *ACM SIGMOD International Conf. on Management of Data*, 1998. 158, 158

22. D. Florescu, A. Levy, and A. O. Mendelzon. Database techniques for the World Wide Web: A survey. *Sigmod Record*, 27(3):59–74, 1998. 151, 155, 155

23. P. Fraternali and P. Paolini. A conceptual model and a tool environment for developing more scalable, dynamic, and customizable Web applications. In *VI Intl. Conference on Extending Database Technology (EDBT'98), Valencia, Spain, March 23–27*. Springer-Verlag, 1998. 158

24. F. Garzotto, P. Paolini, and D. Schwabe. HDM—a model-based approach to hypertext application design. *ACM Transactions on Information Systems*, 11(1):1–26, January 1993. 158

25. R. D. Hackatorn. *Web Farming for the Data Warehouse*. Morgan Kauffman, Los Altos, 1998. 155

26. J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. M. Breunig, and V. Vassalos. Template-based wrappers in the tsimmis system. In *ACM SIGMOD International Conf. on Management of Data*, pages 532–535, 1997. 157

27. B. Inmon. *Building the Data Warehouse*. John Wiley and Sons, New York, 1996. 155

28. T. Isakowitz, M. Bieber, and F. Vitali. Special section on web information systems, introduction. *Communications of the ACM*, 41(7):78–80, July 1998. 151

29. T. Isakowitz, E. A. Stohr, and P. Balasubramanian. RMM: A methodology for structured hypermedia design. *Communications of the ACM*, 58(8):34–44, August 1995. 158

30. D. Konopnicki and O. Shmueli. W3QS: A query system for the World-Wide Web. In *International Conf. on Very Large Data Bases (VLDB'95), Zurich*, pages 54–65, 1995. 156

31. L. Lakshmanan, F. Sadri, and I. N. Subramanian. A declarative language for querying and restructuring the Web. In *6th Intern. Workshop on Research Issues in Data Engineering: Interoperability of Nontraditional Database Systems (RIDE-NDS'96)*, 1996. 156

32. G. Mecca and P. Atzeni. Cut and Paste. *Journal of Computing and System Sciences*, 58(3):453–482, 1999. 157

33. G. Mecca, P. Atzeni, A. Masci, P. Merialdo, and G. Sindoni. From databases to Web-bases: The Araneus experience. Technical Report n. 34-1998, Dipartimento di Informatica e Automazione, Università di Roma Tre, 1987.
http://www.dia.uniroma3.it/Araneus/. 151, 154, 155, 156

34. G. Mecca, P. Merialdo, and P. Atzeni. Araneus in the era of XML. *Data Engineering, IEEE Computer Society*, 22(3), 1999. 159

35. G. Mecca, P. Merialdo, P. Atzeni, and V. Crescenzi. The (Short) ARANEUS Guide to Web-Site Development. In *Proceedings of the Second Workshop on the Web and Databases (WebDB'99) (in conjunction with SIGMOD'99)*.
`http://wwwrocq.inria.fr/~ cluet/webdb99.html`, 1999. 151, 152

36. A. Mendelzon, G. Mihaila, and T. Milo. Querying the World Wide Web. In *First Int. Conf. on Parallel and Distributed Information Systems (PDIS'96)*, 1996. 155, 156
37. Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Eleventh IEEE International Conference on Data Engineering (ICDE'95), Taiwan*, pages 251–260. IEEE Computer Society Press, 1995. 156
38. D. Schwabe and G. Rossi. The Object-Oriented Hypermedia Design Model. *Communications of the ACM*, 58(8):45–46, August 1995. 158
39. A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database systems: Achievements and opportunities. *Communications of the ACM*, 34(10):110–120, October 1991. 150, 151
40. A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database systems: Achievements and opportunities into the 21st century. *ACM SIGMOD Record*, 25(1):52–63, March 1996. 150, 151
41. D. Suciu Ed. Proceedings of the workshop on the management of semistructured data (in conjunction with ACM SIGMOD 1997).
http://www.research.att.com/ suciu/workshop-papers.html. 154
42. J. Widom and S. Ceri. *Active Database Systems*. Morgan Kauffman, Los Altos, 1996. 154

Exploiting Formality in Software Engineering

Juan C. Bicarregui

CLRC, Rutherford Appleton Laboratory
J.C.Bicarregui@rl.ac.uk
www.itd.clrc.ac.uk/Person/J.C.Bicarregui

Abstract. There is undoubtedly a huge gap between the level of formality currently in use in mainstream software engineering and the "best practise" advocated by academics and practised by a small sector of industry involved in critical applications. This paper presents some results of recent research which are building bridges between these two approaches: on the one hand, developing formal methods which are useful to mainstream developers; and on the other, underpinning mainstream methods with formal foundations.

1 Introduction

Formal Methods are now becoming an established technology for software development in several sectors of the industry particularly for systems which are critical with respect to safety or finance. Particular domains where there has been a significant uptake are critical instrumentation and control systems in military and civil avionics systems, terrestrial transportation systems, nuclear control systems, and space systems. Although this is a large industrial market sector, estimated as 3 Billion Euro annually within Europe, there is still much work to be done if formal techniques are to become a major force in mainstream industrial software development.

The formal techniques which have been the subject of the research described in this paper have been available for some years, but technology users have been reluctant to fully adopt what has been offered on the market, as the supplied technologies lacked sufficient track record of use, were perceived as difficult and costly to use, and required highly specialised mathematical expertise in the development team.

In this paper, I give a synopsis of the results of a number of research projects which have made some progress towards providing evidence of the benefit of following the formal approach. This evidence concerns the quality of the delivered system and the cost of its development and is both qualitative and quantitative in nature.

The formal methods considered cover a variety of techniques which can be summarised as follows:

- the use of *Mathematical Models* for specification of software,
- the use of *Proof* for calculation of properties of the specified software,

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 163–180, 1999.

- the use of *Stepwise Refinement* to ensure the preservation of these propeties by the design decisions made during development, and
- the generation of *Test Cases* from specifications to complement formal proof in verification of the implementation.

The formal methods employed in the projects covered by this paper are primarily the "model-oriented" formal methods including:

- VDM [33], the Vienna Development Method originally developed at IBM research laboratories in Vienna during the 1970s, and then consequently at Manchester University, the Technical University of Denmark, RAL and IFAD.
- Z [42], developed at Oxford University during the 1980s, and
- B [1], also emanating from Oxford and then developed in several industrial organisations including BP and GEC Alstrom during the 1980s and 1990s.

This paper is outlines three lines of research undertaken in the author's group at RAL, presents some results from one of those lines, and suggests areas for ongoing further work. The next section identifies the three key lines of research and outlines a number of the research projects undertaken over the last decade. Then Section 3 presents two of these projects in more detail and key results from some from them. The last section describes some ongoing work and draws some conclusions.

2 Three Converging Lines of Research

This section outlines a number of research projects and identifies three areas of research which are key to enabling the uptake of formal methods. The three lines of research are:

1. the *Advancement of Formal Methods* through the development of tools and techniques supporting particular methods,
2. the *Technology Transfer of Formal Methods* through their application in developments in collaboration with industry, and
3. the *Formalisation of Industrial Methods*, in particular diagrammatic methods for Object Oriented Analysis and Design.

These three lines of research contribute respectively to the practical feasibility of using formal methods in real-life software development; to the body of evidence for the costs and benefits of adopting them; and to the reduction of the level of specialisation required by the personel employing them.

The projects covered are summarised in the following sections.

2.1 Advancement of Formal Methods

The first line of research is concerned with the advancement of formal methods themselves and the devlopment of technology to support their use. These projects developed tools and standards for formal methods, both prerequisites to the industrial uptake of a technology.

The Mural project (1986-1990) developed a VDM specification support tool and proof assistant. Working with Cliff Jones' group at Manchester University, this project produced a tool supporting VDM developments and a generic proof assistant supporting the construction of proofs arising in those developments [23,25,24]. The resulting tool was among the first to a use graphical user interface to a theorem proving assistant and has since inspired a number of projects involving user interface design for theorem provers.

Proof in VDM. The Mural project led to the definition of an axiomatic semantics for VDM and a "Practitioner's Guide" to proof using that semantics [15]. A number of case studies in proof using that axiomatic semantics were also developed [7]. It also led to a line of reserach into the use of read and write frames in operation decomposition [9,10,8].

The ZIP project was a collaboration between British Aerospace, British Petroleum, IBM, Logica, Praxis, the University of Oxford and the Rutherford Appleton Laboratory. This project developed a semantics and proof theory for Z, which forms the basis of the current Draft of the British and ISO Standards for Z.

Verification Techniques for LOTOS (1990-1992) was a study of the theory and practise of the verification of specifications in the ISO standard LOTOS language. RAL's contribution to this project was in the development of the ERIL term rewriting formal reasoning system [37] and in undertaking a case study which used LOTOS to specify parts of the graphics standard GKS [29]. The other partners in the project were Royal Holloway and Bedford New College, Glasgow University, and British Telecommunications.

2.2 Technology Transfer of Formal Methods

The second line of research concerns the technology transfer of formal techniques into industrial practice. This has been achieved through three industrial collaborative projects which have employed and assessed the VDM and B methods. These projects provide evidence of the cost/benefit of the use of formal methods necessary to reduce the risk of their uptake.

The B User Trials project (1992-1995) was the first UK industrial project involving the B-Toolkit. It was a collaborative project between RAL, Lloyds Register of Shipping, Program Validation Limited and the Royal Military College of Science and played a major part in bringing the B-Toolkit up to industrial quality. RAL undertook two case studies [22,41] comparing B with VDM and Z respectively, and investigated the methodology employed in the construction of proofs [19,26]. A summary of the project appears in [11].

The MaFMeth project (1994-1995) was the first project to bring VDM and B together to exploit their different strengths. A collaboration with Bull Information Systems, it was "application experiment" assessing a methodology covering the whole life cycle combining the use of VDM for early lifecycle development with B for refinement and code generation. It demonstrated the commercial viability of the use of formal methods by collecting quantitative evidence of the benefits in terms of fewer faults made in development and early detection of faults. Qualitative experience is reported in [14] and a quantitative analysis of the results in [13]. In particular it was found that animation, test case generation and proof are all cost-effective ways to find faults in formal texts [12].

The Spectrum project (1997) was a feasibility study into the commercial viability of intergating the VDM-Toolbox and B-Toolkit. The evaluation was undertaken from three perspectives: the industrial benefit of using the combined tool, the technical feasibility of the combination of the two tools and the commercial case for the development of a combined tool. RAL's partners were GEC Marconi Avionics, Dassault Electronique, Space Software Italia, the Commissariat à l'Energie Atomique, the Institute of Applied Computer Science (IFAD) and B-Core UK Ltd.

Further details of the MaFMeth and SPECTRUM projects are given in the Section 3.

2.3 Formalisation of Industrial Methods

These projects concerned providing some formal underpinning to techniques already established in industry, thus providing a low-cost entry route to the uptake of formal methods.

The TORUS project sought to raise the level of reuse in industrial process control projects, by defining and deploying reuse based work processes together with supporting tools, thus improving the efficiency over the whole design lifecycle. the project employed the ISO standard data modelling language STEP-EXPRESS. As part of the work, we undertook investigations into the formalisation of this langauge, and made suggestions for clarifications to its semantics [20,18]. The project was in collaboration with Cegelec Projects Ltd., FLS Automation, and Alcatel ISR.

Formal Underpinning for Object technology (FUOT) This research with Imperial College London and the University of Brighton undertook the formalisation of the conceptual basis of diagrammatic Object Oriented Analysis and Design notations such as UML using the "Object Calculus" as a semantic framework [17]. It also analysed the notations and typical development steps in order to suggest improvements to make these techniques more sound scientifically. The work also raised some issues about subsystems which are the basis of ongoing research [16] (see Section 4.2).

3 Results of Technology Transfer Projects

This section presents in more detail one strand of research undertaken in the second of these research lines: the technology transfer of formal methods. It describes two projects, MaFMeth and Spectrum which have both been concerned with the assessment of the use of a combination of the VDM and B Methods. The next section presents some key results from those projects.

3.1 Heterogeneous Development Using VDM and B

VDM and B are two of the most industrially used formal methods. Both are model-oriented methods for the development of sequential systems based on first order calculi and set theory. Both have a set of proof rules defined for formal verification and validation. Both have a formal semantics: for B this is defined in terms of weakest preconditions, for VDM it is denotational. Both have been used for a variety of applications and are supported by commercial toolkits.

VDM – The Vienna Development Method VDM's origins lie in the definition of programming language semantics in the 1970s, but for many years it has been used in systems development generally [33], and there is now an ISO standard [32] of the specification language VDM-SL. It has a rich set of data type constructors, augmented by invariant predicates. Functions and state-transforming operations can be defined explicitly using a large expression and statement language or implicitly in terms of precondition and postcondition predicates.

The VDM-SL standard includes a denotational semantics [36]. The semantics is based on the three-valued Logic of Partial Functions [34] which explicitly deals with definedness of expressions and requires the demonstration of well-typing for the substitution of equals. In particular, the strong type system supports static detection of many well-formedness errors. A published proof theory, described in [33] and in greater detail in [15], supports the validation of VDM-SL specifications through the discharge of proof obligations.

One area of weakness in VDM relative to B is its lack of generally agreed large-scale structuring. The standard contains a "informative annex" describing several alternative approaches to modules, including one implemented within the IFAD Toolbox, and other structuring proposals exist [31].

The IFAD VDM-SL Toolbox [30] is an industrial strength commercially available tool which supports the ISO VDM-SL notation. The Toolbox includes a syntax checker, static semantic checker, and a pretty printer generating LaTeX output. In addition, it contains a debugger, an interpreter, and a C++ code generator. It is also possible to perform test coverage analysis of specifications. An earlier tool for VDM, VDM Through Pictures [27] developed by Bull Information Systems, which supported the generation of VDM "skeletons" from Entity-Relationship style diagrams was used in the MaFMeth project.

Areas to which VDM has recently been applied include railway interlocking systems, ammunition control systems, semantics of data flow diagrams, message authentication algorithms, relational database systems and medical information systems. A directory of VDM usage examples is available [45].

The B Method The B-Method [1] represents one of the most comprehensive formal methods currently being promoted as appropriate for commercial use. A development of VDM and Z, Jean-Raymond Abrial originated **B** whilst at the Programming Research Group at Oxford University in the early 1980s, and subsequently at British Petroleum Research (BP) and DIGILOG. Supported forms are now commercially available from B Core Ltd. and Steria Technologies de l'Information.

The B-method employs the Abstract Machine Notation (B-AMN) which uses a notion of *generalised substitution* to represent state transformations, a style of specifying operations which is more "natural" to the programmer than the pre/post predicates of VDM. The B-method also has powerful structuring mechanisms based on a notion of *Abstract Machine* which offers data encapsulation allowing modular design and development of systems.

B's underlying semantics is grounded in weakest preconditions over untyped set theory and classical logic; the type system is correspondingly weak, and the distinction between type-checking and proof is blurred.

The B-Toolkit [6], developed by BP and subsequently by B-Core UK Ltd focuses on rigorous/formal design by supporting refinement from abstract specification through to imperative code. Tools exist for supporting static analysis (type-checking), dynamic analysis (animation), design documentation, proof of refinement and code generation. Another support system for B is Atelier-B [4], provided by Steria Mediterranée which allows similar functionality to the B-Core Toolkit.

Examples of the use of B include the development of communication security protocols, subway speed control mechanisms, railways signalling, executable database programs and IBM's CICS product. A directory of B is maintained at [5].

Co-use of VDM and B Although VDM and B have the same expressive power in theory, a comparison undertaken during the B User Trials project observed [22] that VDM encourages a style of specification where implicit invariants and explicit frames are employed with postconditions to describe operations as abstractly as possible, whereas the representation of operations with explicit invariants and implicit frames employed in B encourages overspecification and the introduction of implementation bias reducing possible non-determinism. This difference arises from the different focus of the two methods and has led to the development of different functionality in the supported forms of the methods. Figure 1 and Table 1 show the different phases of the lifecycle favoured by the two methods and the complementary features currently provided by each toolkit.

3.2 MaFMeth: The Measurement and Analysis of a Formal Method

The combination of VDM and B was first explored by RAL and Bull Information Systems in the MaFMeth Project [13,12]. This project developed part of a transaction management system using VDM for the initial design and analysis, and then translating into B for development and code generation.

The formal development was part of the second release of an application integration product of the type often known as "middleware" which allows applications to communicate in a number of ways via a single application programming

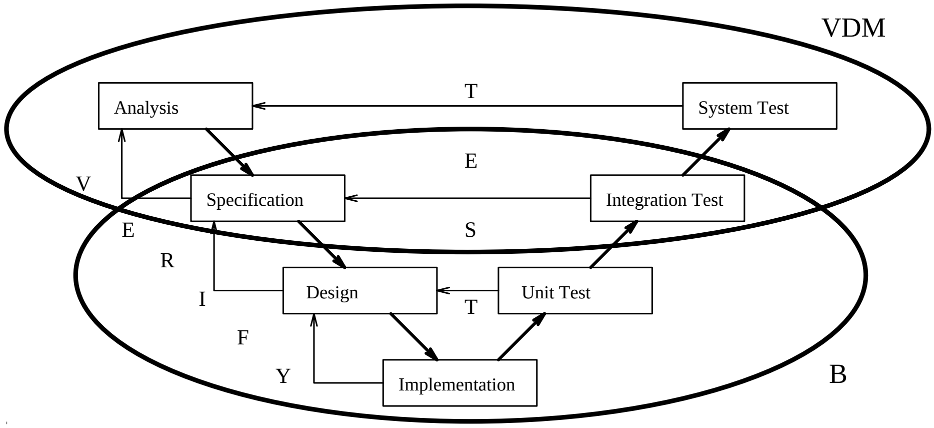


Fig. 1. The lifecycle identified for heterogeneous development using VDM & B

Table 1. The complementary functionality of the VDM and B Toolkits

Task	IFAD VDM-SL Toolbox	B-Toolkit
Requirements capture	×	×
Visualisation	×	×
Abstract Specification	√	×
Type checking	√	∼
Prototype code generation	√	∼
Test coverage	√	×
Animation/Execution	√	√
Modularity	∼	√
Refinement	×	√
Proof	×	√
Final Code generation	×	√
Design documentation	×	√
Version Cntl/Config Mgmt	×	∼

Key
√ = good support
∼ = some support
× = no support

interface. Its primary function is to provide distributed, multi-platform, inter-application message handling services involving message routing, storage, transformation and enrichment transparently to the applications. The component of the system which was developed formally monitors the status and contents of the message queues and allows individual messages to be updated when required.

The project was undertaken in a conventional system software department with a development process certified as ISO9001 (TickIt) [44] compliant for its quality management system and operating at a point close to level 3 of the SEI Capability Maturity Model [40]. The development process adopted in this project was influenced by the desire to assess a variety of formal techniques covering as much of the development life cycle as possible, and the requirement that the resulting code had to be closely integrated with code developed by other methods.

The decision to employ a combination of VDM and B was motivated by the complementary facilities offered by the two toolkits and previous experience [22] which had shown the complementary strengths of the VDM and B methods. Three formal specifications were produced. The first, most abstract, specification was developed in VDM using VDM through Pictures. This was translated by hand into B Abstract Machine Notation, in order to conduct the first and second specification decomposition with the B-Toolkit, the result of which was then used to automatically generate C code.

Three forms of analysis were undertaken for validation and verification. *Animation* was used to validate the design during development, whereas post facto verification was undertaken using *test cases* and *proof obligations* which were generated from the specifications.

Measurements relating to these activities were taken in order to compare the formal development process with the conventional one used in that department, and to compare the relative effectiveness of the various stages of the formal process. For the former, the results of a number of development projects, all producing sub-products with similar characteristics, were compared using the department's existing programme of metrics. For the latter, faults were classified according to the development stage at which they were discovered and the stage at which they were introduced.

3.3 SPECTRUM: A Step Toward a Unified Method

Following MaFMeth, the SPECTRUM project also assessed the benefit of the combined use of VDM and B within a single development process. The project was a collaboration with four user partners in considering application domains:

- GEC Marconi Avionics considering avionics systems,
- Dassault Electronique considering terrestrial transport embedded control,
- Space Software Italia considering satellite communication control, and
- Commissariat à l'Energie Atomique considering nuclear plant control;

and two technology suppliers:

- the Institute of Applied Computer Science (IFAD) suppliers of the VDM-Tools, and
- B-Core UK Ltd. suppliers of the B-Toolkit.

The Rutherford Appleton Laboratory provided two roles:

- Expertese in VDM and B and their combination, and
- Neutral Project Coordination.

The SPECTRUM project had three objectives:

- to assess the cost-effectiveness for the user partners, of a development process employing an integration of the VDM and B formal technologies,

- to determine the technical feasibility of this integration; and
- to investigate the commercial potential for the tool suppliers of the integration of supported forms of VDM and B.

The first objective was addressed through the development of user scenarios exploring the utility of the various functions available from the two toolkits and the advantages of employing them in terms of development cost and product quality. The scenarios were in the application domains of avionics systems and terrestrial transport embedded control. They were also reviewed from the perspectives of satellite communication control and nuclear power plant control. Further details of the case studies can be found in [28,43,3].

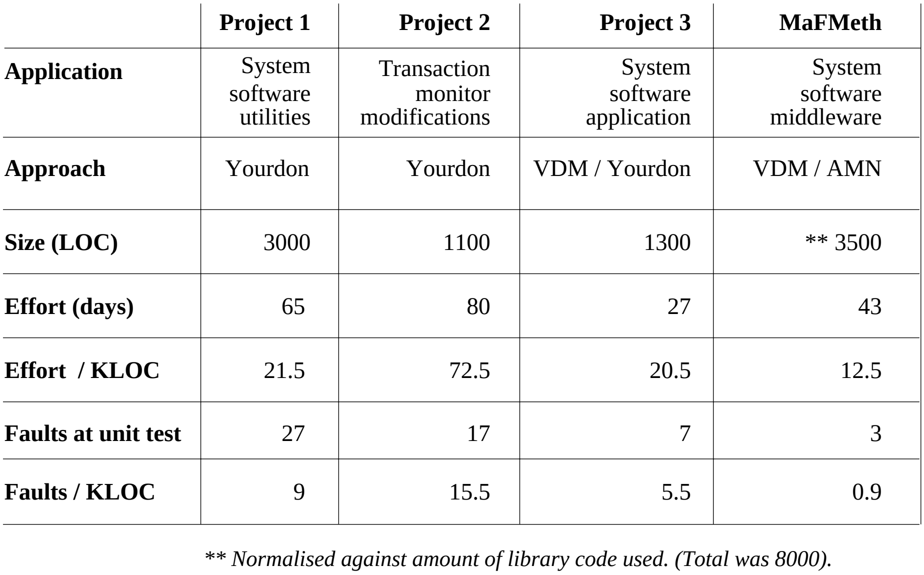
The second objective was address through investigations into the feasibility of automating support for the translations between VDM and B. An approach to translation requiring the analysis of usage of types in the flat VDM specification in order to synthesise a useful decomposition into modules in B was proposed. Further details of the translations can be found in [38,21]. This work formed a key input to the VDM+B project described in Section 4.1.

To address the third objective, the project assessed the commercial case for the provision of required functionality. It explored the overall cost-benefits of introducing the proposed method and the commercial viability of the proposed tool integration. It prepared a detailed market analysis and pricing policy for the combined tools. The results of this work are considered to be commercial in confidence.

3.4 Evidence of Error Reduction

The MaFMeth project undertook measurement and analysis of the total number of faults introduced and found during development. Despite the use of two notations and the lack of integrated tool support, quantitative analysis of the faults found at unit test shows the approach to be very effective both in cost and quality. Figure 2 compares data from this project with three others undertaken by the user partner using structured design. The four projects were all developed in the same environment over a period of about 3 years and all used a similar development process apart from the technology involved. All projects were undertaken by engineers from the same development group and all were fragments of much larger developments. Similar testing procedures, based on manual identification of tests, were followed in each case. All, bar project 2, were new developments, whereas project 2 was a complex modification to an already heavily maintained system software component (hence, perhaps, the low productivity and quality of that development). None of the effort figures include the learning and technology transfer time which is inevitable in applying new approaches.

The LOC figure (Lines of Code) is clearly central to the metrics and, for projects 1 to 3, refers to C language statements. For MaFMeth, in all 8000 lines of code were generated. However much of this arose from library components. The figure of 3500 lines of code is the developer's estimate of the amount of code



	Project 1	Project 2	Project 3	MaFMeth
Application	System software utilities	Transaction monitor modifications	System software application	System software middleware
Approach	Yourdon	Yourdon	VDM / Yourdon	VDM / AMN
Size (LOC)	3000	1100	1300	** 3500
Effort (days)	65	80	27	43
Effort / KLOC	21.5	72.5	20.5	12.5
Faults at unit test	27	17	7	3
Faults / KLOC	9	15.5	5.5	0.9

*** Normalised against amount of library code used. (Total was 8000).*

Fig. 2. Comparison of numbers of faults found

that would have been produced to implement the same functionality without attempting any reuse. In fact, 1200 lines of implementation level B notation were produced to generate the final C code.

The figures show that the MaFMeth project produced, on average, more code per day than any of the previous projects. Of course, this result must be tempered by the different application areas and the possible inaccuracy in the estimate of the equivalent number of lines of code. However, the improvement of nearly 100 % is noteworthy.

Even more significant are the results concerning the number of faults at unit test. The unit testing used aimed at 100 % functional black box test coverage and 100 % branch level white box coverage. This was achieved by identifying test cases using techniques including equivalence partitioning, boundary value analysis and a judicious amount of error guessing! The MaFMeth project produced less than 20 % of the faults of the next best project.

No attempt was made to moderate the effectiveness of fault finding by the sev the faults found. Such an analysis should contribute to an estimate of the cost-effectiveness of each activity.

Unfortunately, no figures for faults found during validation testing and customer use are available.

3.5 Evidence of Cost Reduction

Further evidence of cost reduction and comarison with costs for other development approaches was provided by GEC Marconi in SPECTRUM. Figure 3 shows an abstracted graph plotting the expected cost of using different formal methods in three projects over recent years.

The two horizontal lines represent the expected cost of employing their informal development process for saftey critical and non-safety critical system components.

The graph shows the falling cost of formal methods over recent years and in particular emphasises that the cost is now competative for safety critical systems and approaching viability for non-critical developments.

The vertical bars associated with the data points represent the high margin for error in prodictions made from a few small projects.

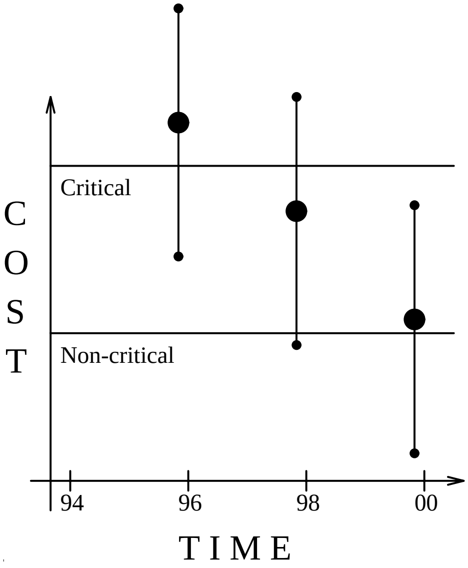


Fig. 3. Comparison of numbers of faults found

3.6 Relative Benefit of Formal Activities

The MafMeth project also undertook an analysis of the relative cost/benefits of each develepment activity. For these puposes the development process was divided into 13 activities, with varying degrees of tool support. These are depicted in Figure 4. The distribution of effort by activity is shown in Figure 5. Some activities, for example the initial B specification and its animation, are grouped together as they were carried out simultaneously and no separate effort figures were kept.

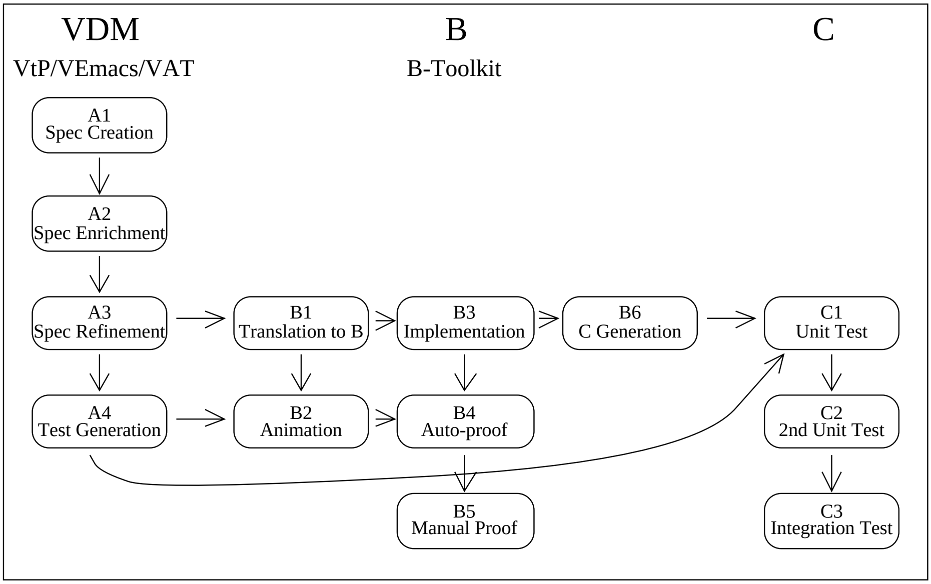


Fig. 4. Development activities identified in the MaFMeth project

As might have been expected, the bulk of the design effort was in the main development in B. A substantial component was also expended on the early specifications in VDM. Very little effort was required during the testing stage.

The faults found can be plotted against these efforts as a histogram with the width of columns representing the relative effort expended in each stage (Figure 6). However, when inspecting this it must be remembered that some stages involved development whereas others purely involved review. For stages B1-2, one cannot assess how much effort was expended in finding faults through animation and how much on development, but if one assumes that approximately one half of this effort was spent on each activity, then the dotted line applies.

Note how the most efficient fault finding occurs during test generation, animation and proof. Although this can perhaps be attributed to the fact that most faults were found before testing occurred, the test generation and proof stages allow a different perspective on the specification and highlight problems which might otherwise be invisible to the developer.

4 Ongoing Work and Conclusions

This section outlines two current projects which are bringing together the three lines of research described above. The VDM+B project building on lines 1 and 2, the advancement and technology transfer of formal methods; and the Subsystems

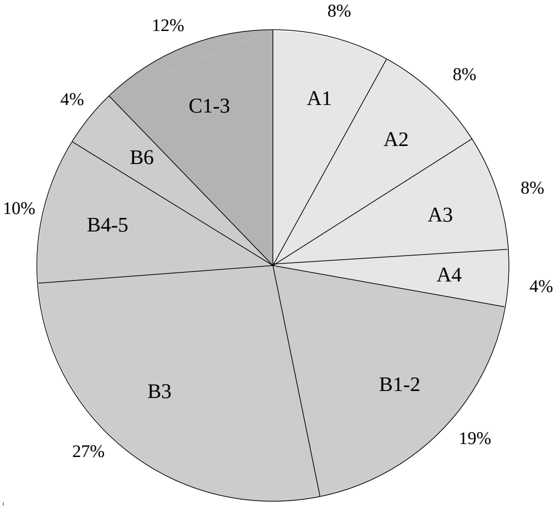


Fig. 5. Effort expended by each project stage

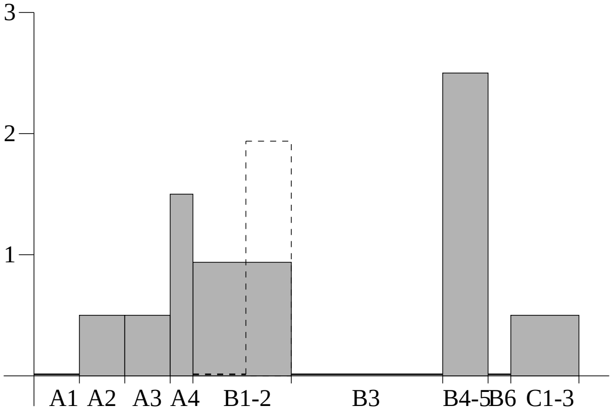


Fig. 6. Faults found per day by project stages

project building on lines 1 and 3, the advancement of formal and industrial methods, it then draws some overall conclusions.

4.1 The Integration of Two Industrially Relevant Formal Methods (VDM+B)

The VDM+B project (1998-2001) is the third project on VDM and B integration building on the results of the MaFMeth and SPECTRUM projects. In recognition of the pragmatic nature of the earlier work, the goal of this project is establish a formal foundation of heterogeneous development in VDM and B.

An obvious point of concern is the foundational differences in the languages. VDM is based on the 3-valued Logic of Partial Functions (LPF) whereas B is based on classical First Order Predicate Calculus. Work on developing proof support for VDM [2] has shown that in a framework with dependent types, such as PVS [39], most specifications which employ partial functions for their expressivity can be directly translated to functions which are total over a subdomain. The remaining uses of partiality represent a particular form of lazy concurrent disjunction which is built into LPF but not available in B.

Although the two notations are founded on a different logic, the proof rules in the B-Toolkit do have a flavour more akin to those of VDM where typing hypotheses are used as guards to the expression construct introduction and elimination rules. In the absence of a standard form for proofs that would enable proofs developed in one system to be checked with another, it is important for the certification of formal developments to be able to "second source" the theorem proving capability. This will allow proof support to be developed in a number of systems and contribute to the certification of theorem proving capability for use in safety critical systems. Current support for the languages has not been certified in this way.

A further area of difference is higher level modular structuring. Several approaches to modularisastion exist for VDM. The VDM standard language, VDM-SL, has no structuring mechanism although a form of modularisation is given as an informative annex, the IFAD VDM Toolbox supports a simple form of modules and VDM++ [35] has an object-oriented notion of structuring based on classes. On the other hand, the ability to incrementally present a specification is central to B where implementations can be constructed in a structured way by composing implementations of separate components.

Thus transformations between structured specifications in the two formalisms should, in some sense, preserve the locality of information. For example, in moving from a single module of VDM where the structure is based around a hierarchical definition of record types, we would hope to achieve a B specification which used machines to mirror the structure of the records. The danger is that in "coding up" such a complex refinement into the translation we risk the soundness of the translation. One possible approach [38] is for the translation to result in two levels of B specification and a refinement between them. In this way the translation is kept simple, whilst the complexity of the refinement is localised within the one formalism and hence more amenable to verification.

4.2 Objects, Associations and Subsystems: A Hierarchical Approach to Encapsulation

The Subsystems project (1999-2002) arose from observations made in the Formal Underpinning of Object Technology project. That project observed that although subtyping and inheritance provide a hierarchical means of classification of objects, the class-instance paradigm is essentially flat and does not directly support the nesting of objects within objects. This led us to propose a notion of subsystem which generalises the class-instance-based concept of object, yielding an approach to system-specification employing object-like encapsulation in a nested hierarchy of components [16].

The strength of these subsystems lies in generalising key features of the success of object orientation. Objects provide a simple yet powerful basis for modularity through encapsulation. Aggregation of attributes in objects, and objects in associations, provides a basis for data-encapsulation; object identifiers globally identify instances and give an implicit indirection which distinguishes attributes which are themselves objects from attributes which are pure values. Objects can also provide a basis for establishing non-interference in concurrent implementations. On this basis, it seems that the OO approach would benefit from an old idea: hierarchical structuring.

In [16], we observed that the compositional interpretation of object-oriented designs requires the identification of theories intermediate between those of the constituent classes and associations and that of the entire system; and, how many constructions are naturally interpreted in theories corresponding to identified parts of the overall system. This project will investigate subsystems as first-class objects in OO system description achieving a hierarchical form of object-orientation.

4.3 Conclusions

We have presented some key results of two projects measuring the costs and benefits of the formal approach. This evidence indicates that formal methods are currently cost-neutral for safety critical systems and deliver higher quality than alternatives. The evidence contributes to the case upon which the adoption of formal techniques should spread beyond those applications domains where formal techniques and mandated or highly recommended such as defense and transport (c.f. UK Def-Stan 00-55 and French RATP requirements) to other safety and financially critical applications.

We have indicated the falling cost of formal methods which are now becoming competative with other methods used in non-critical systems development. Further cost reduction, in particular below the cost of systems development for non-critical systems, will increase the market enormously.

However, conclusions drawn from these projects should be moderated by the small size the developments and the fact that the development teams were also small and staffed by self-selected individuals who, being keen to make a success of the experiments, were perhaps better motivated than average. It would not be wise therefore to extrapolate these results to larger projects.

Despite these qualifications, there is some evidence in these results in favour of formal methods. Faults are inevitable and their detection is aided by formalisation. It seems that any analysis, whether animation, proof obligation generation, proof, or testing, is worthwhile. These activities are only possible once the objects involved are formalised.

The three lines of research described provide three-legs of support for industrial strength formal methods: improved methods and support for them; evidence of the costs and benefits of their use; and formal underpinning of established industrial methods.

This work has contributed to an accumulation of evidence for the benefits of formal methods. It has raised awareness of the need to gather such evidence for larger projects and has demonstrated some techniques for doing so.

References

1. J.-R. Abrial, *The B-Book: Assigning Programs to Meaning*, Cambridge University Press, ISBN 0-521-49619, 1996. 164, 167
2. S. Agerholm, Translating Specifications in VDM-SL into PVS, in Higher Order Logic Theorem and Its Applications: 9th International, LNCS 1125, Springer Verlag, 1996. 176
3. S. Agerholm, P-J. Lecoeur and E. Reichert., *Formal Specification and Validation at Work: A Case Study using VDM-SL* Proceedings of Second Workshop on Formal Methods in Software Practice, Florida, ACM, March 1998. 171
4. Atelier B, Steria Mediterranée, The B Page URL, http://www.atelierb.societe.com/PAGE_B/uk/bhomepg.htm 168
5. The B method (virtual library page) http://www.comlab.ox.ac.uk/archive/formal-methods/b.html. 168
6. B-Core (UK) Ltd, *B-Toolkit User's Manual, Version 3.0*, 1996. For details, contact Ib Sorensen, B Core (UK) Ltd, Magdalen Centre, Robert Robinson Avenue, The Oxford Science Park, Oxford OX4 4GA. Tel: +44 865 784520. Email: `Ib.Sorensen@comlab.ox.ac.uk`, WWW: http://www.b-core.com/ 168
7. J. Bicarregui (Ed.) *Proof in VDM: Case Studies*, Springer-Verlag, FACIT series, 1998. 165
8. J. C. Bicarregui. *Intra-Modular Structuring in Model-Oriented Specification: Expressing Non-Interference with Read and Write Frames.* Ph.D. Thesis, University of Manchester (UMCS-95-10-1). 165
9. J. C. Bicarregui. *Operation Semantics with Read and Write Frames.* Proceedings of the 6th Refinement Workshop, David Till (Ed.), Springer Verlag. 165
10. *Algorithm Refinement with Read and Write Frames.* J.C. Bicarregui. Proceeding of Formal Methods Europe '93, Woodcock and Larsen (Eds), LNCS 670, Springer-Verlag. 165
11. J. C. Bicarregui et al.. *Formal Methods Into Practice: case studies in the application of the B Method.* **I.E.E. proceedings software engineering, Vol. 144, No. 2**, 1997. 165
12. J. C. Bicarregui, J. Dick, B. Matthews, E. Woods. *Making the most of formal specification through Animation, Testing and Proof.* **Science of Computer Programming, Vol. 29.** (1997), Elsevier Science. 166, 168
13. J. C. Bicarregui, J. Dick, E. Woods. *Quantitative Analysis of an Application of Formal Methods.* Proceeding of FME'96, Third International Symposium of Formal Methods Europe, LNCS, Springer-Verlag. 166, 168
14. J. C. Bicarregui, J. Dick, E. Woods. *Supporting the length of formal development: from diagrams to VDM to B to C.* Proceedings of 7th International Conference on: "Putting into practice method and tools for information system design", Nantes (France), October 1995, IUT de Nantes, H. Habrias (Ed.) ISBN 2-906082-19-8. 166

15. *Proof in VDM – A Practitioner's Guide.* J. C. Bicarregui, J.S. Fitzgerald, P. A. Lindsay, R. Moore and B. Ritchie. ISBN 3-540-19813-X, FACIT, Springer-Verlag 165, 167
16. J. C. Bicarregui, K. C. Lano and T. S. E. Maibaum, *Objects, Associations and Subsystems: a hierarchical approach to encapsulation*, Proceedings of ECOOP'97, 11th European Conference on Object-Oriented Programming, Jyväskyla, Finland, June 1997. 166, 177, 177
17. *Towards a Compositional Interpretation of Object Diagrams.* J. C. Bicarregui, K. C. Lano and T. S. E. Maibaum, in "Algorithmic Languages and Calculi", Proceedings of IFIP TC2 working conference, Strassbourg, Feb. 1997, Bird and Meertens (Eds.), Chapman and Hall. 0-412-82050-1, September 1997. 166
18. J. C. Bicarregui, B. M. Matthews, *The specification and proof of an EXPRESS to SQL "compiler"*, in Proof in VDM: Case Studies, J. C. Bicarregui (Ed.), FACIT series, Springer-Verlag, 1998. 166
19. J. C. Bicarregui, B. Matthews. *Formal Methods in Practice: a comparison of two support systems for proof.* SOFSEM'95: Theory and Practice of Informatics, Bartošek et al. (Eds.), LNCS 1012, Springer-Verlag. 165
20. J. C. Bicarregui, B. M. Matthews *Formal perspectives on an Object-Based Modelling Language* Proceedings of the 6th EXPRESS User Group Conference, Toronto, Canada. IEEE Computer Society, ISBN 0-8186-8641-3, 1996. 166
21. *Investigating the integration of two formal methods*, Juan Bicarregui, Brian Matthews, Brian Ritchie, and Sten Agerholm, *Formal Aspects of Computing,* Vol. 10. pp. 532–549, Springer-Verlag, December 1998. 171
22. *Invariants, Frames and Postconditions: a comparison of the VDM and B notations.* J. C. Bicarregui and B. Ritchie. Proceeding of Formal Methods Europe '93, Woodcock and Larsen (Eds.), LNCS 670, Springer-Verlag. b. 1995. 165, 168, 170
23. *Reasoning about VDM Developments using the VDM Support Tool in Mural.* J. C. Bicarregui and B. Ritchie. Proceeding of VDM 91, Prehn and Toetenel (Eds), LNCS 552, Springer-Verlag. 165
24. *Supporting Formal Software Development* and *The Mural VDM Support Tool.* J. C. Bicarregui and B. Ritchie. in "Mural, A Formal Development Support System." Jones, C. B. et al. Springer-Verlag, ISBN 3-540-19651-X. 165
25. *Providing Support for the Formal Development of Software.* J. C. Bicarregui and B. Ritchie. Proceedings of the First International Conference on Systems Development Environments and Factories. Madhavji, Schafer and Weber (Eds.), Pitman. 165
26. *Experiences with Proof in a Formal Development*, D. Clutterbuck, J. C. Bicarregui and B. M. Matthews, Proceeding of 1st International Conference on B, Institut de Recherche en Informatique de Nantes, France, November 1996. 165
27. A. J. J. Dick and J. Loubersac. *A Visual Approach to VDM: Entity-Structure Diagrams,* Technical Report DE/DRPA/91001, Bull, 68, Route de Versailles, 78430 Louveciennes (France), 1991. 167
28. J. Draper (Ed.) *Industrial benefits of the SPECTRUM approach* SPECTRUM Project External Deliverable 1.3, 1997. See http://www.itd.clrc.ac.uk/Activity/SPECTRUM. 171
29. D. A. Duce, F. Paterno, *Lotos Description of GKS-R Functionality*, Formal Methods in Computer Graphics Workshop Eurographics Association, 1991. 165
30. R. Elmstrøm, P. G. Larsen, and P. B. Lassen, *The IFAD VDM-SL Toolbox: A Practical Approach to Formal Specifications*, ACM Sigplan Notices 29(9), pp. 77–80, 1994. For more information see http://www.ifad.dk/ 167

31. J. S. Fitzgerald, *Modularity in Model-Oriented Specification and its Interaction with Formal Reasoning.* Ph.D. Thesis, University of Manchester, 1991. 167
32. ISO *Information Technology – Programming Languages – Vienna Development Method-Specification Language. Part 1: Base Language.* ISO 13817-1, 1996. 167
33. C. B. Jones, *Systematic Software Development using VDM.* Prentice Hall, second edition, 1990. 164, 167, 167
34. C. B. Jones and C.A. Middelburg, *A Typed Logic of Partial Functions Reconstructed Classically,* Acta Informatica, 1994. 167
35. K. Lano, S. Goldsack, Integrated Formal and Object-oriented Methods: The VDM++ Approach', 2nd Methods Integration Workshop, Leeds Metropolitan University, April 1996; 176
36. P. G. Larsen and W. Pawlowski, *The formal Semantics of ISO VDM-SL,* Computer Standards and Interfaces, Volume 17, numbers 5–6, 1995. 167
37. B. M. Matthews, *MERILL: An Equational Reasoning System in Standard ML*, 5th Int. Conf. on Rewriting Techniques and Applications p. 414–445 C. Kirchner (Eds.), LNCS 690, Springer-Verlag, 1993. 165
38. *Synthesising structure from flat specifications*, B. Matthews, B. Ritchie, and J. Bicarregui, Proc. of the 2nd International B Conference, Montpellier, France, April 22–24, 1998. Springer Verlag, LNCS 1393. 171, 176
39. S. Owre et al. PVS: Combining Specification, Proof Checking, and Model Checking, Computer-Aided Verification, CAV '96, Rajeev et al. (Eds.) Springer-Verlag LNCS 1102, 1996. 176
40. M. C. Paulk, W. Curtis, M. B. Chrissis, C. V. Weber, *Capability Maturity Model for Software*, Version 1.1, Carnegie Mellon University Software Engineering Institute Technical Report, CME/SEI-93-TR-24, February 1993. 169
41. *Experiences in Using the Abstract Machine Notation in a GKS Case Study.* B. Ritchie, J. C. Bicarregui and H. Haughton. Proceeding of Formal Methods Europe '94, Naftalin, Denvir and Bertran (Eds.), LNCS 873, Springer-Verlag. 165
42. J. M. Spivey. *The Z Notation, A Reference Manual.* Prentice Hall, ISBN 0-13-983768-X, 1987. 164
43. H. Treharne, J. Draper and S. Schneider, *Test Case Preparation Using a Prototype*, Proc. of 2nd Int. B Conf. B'98: Recent Advances in the Development and use of the B Method, LNCS 1393, Springer Verlag, April 1998. 171
44. U. K. Department of Trade and Industry, *TickIT: Guide to Software Quality Management, System Construction and Certification using ISO9001/EN29001/BS5750 Part 1*, TickIT Project Office, 68 Newman Street, London, W1A 4SE, UK, February 1992. 169
45. The VDM examples repository: http://www.ifad.dk/examples/examples.html. 167

Biomolecular Computing and Programming
(Extended Abstract)

Max H. Garzon, Russell J. Deaton, and The Molecular Computing Group

The University of Memphis, Memphis TN, 38152
www.msci.memphis.edu/~garzonm/mcg.html

Abstract. Molecular computing is a discipline that aims at harnessing individual molecules at nanoscales to perform computations. The best studied molecules for this purpose to date have been DNA and bacteriorhodopsin. Biomolecular computing allows one to realistically entertain, for the first time in history, the possibility of exploiting the massive parallelism at nanoscales for computational power. This talk will discuss major achievements to date, both experimental and theoretical, as well as challenges and major potential advances in the immediate future.

Early ideas of molecular computing attempted to emulate conventional electronic implementations in other media, e.g., implementing Boolean gates in a variety of ways. A fundamental breakthrough characteristic of a new era was made by Adleman's 1994 paper in *Science*, where he reports an experiment performed with molecules of fundamental importance for life, DNA (deoxyribonucleic acid) molecules, to solve a computational problem known to be difficult for ordinary computers, such as the HAMILTONIAN PATH PROBLEM (HPP). There is good evidence, however, that several million years earlier and unknown to all of us, the ciliated protozoa *oxytricha nova* and *oxytricha trifallax* of the genus *oxytricha* solved a problem similar to HPP while unscrambling genes as part of their reproductive cycle. It has been argued that other molecules such as proteins and artificially engineered rybozymes may serve biological and computational functions much better. An older alternative to DNA molecules that support optical computing is the protein *bacteriorhodopsin*, which contains the light sensitive *rhodopsin* present in vertebrate retinas. The light activated state switching property has been used in combination with lasers to create a storage medium for optical computer memories that is almost in the commercial stage now. The possibility exists that it might become a core memory for a molecular computer. Although certainly involving amino acids at the protein-binding sites, this type of computation is more passive than Adleman's type. 'Molecular computing' thus essentially means today DNA- and RNA-based computing.

Adleman's seminal idea insight was to carefully arrange a set of DNA molecules so that the chemistry that they naturally follow would perform the brunt of the computational process. The key operations in this chemistry are sticking operations that allow the basic nucleotides of nucleic acids to form larger structures through the processes of *ligation* and *hybridization*. The first DNA-based molecular computation is summarized in Fig. 1. Specifically, Adleman

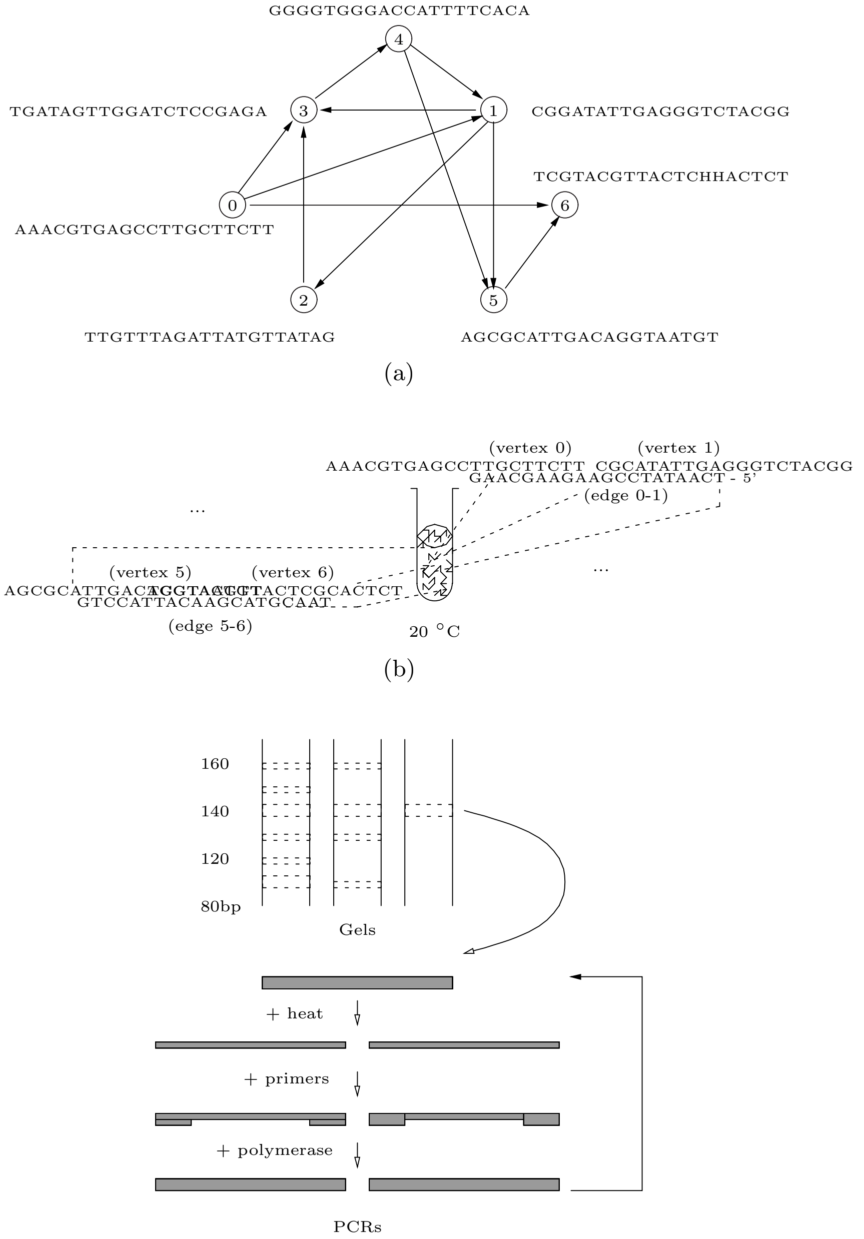
J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 181–188, 1999.

assigned (well chosen) unique single-stranded molecules to represent the vertices, used Watson-Crick complements of the corresponding halves to represent edges joining two vertices, and synthesized a picomol of each of the 21 resulting molecules for the graph in Fig. 1(a). Taking advantage of the fact that molecular biologists have developed an impressive array of technology to manipulate DNA, he designed a molecular protocol (one would say *algorithm* in computer science) that enabled the molecules to stick together in essentially all possible ways. In the situation illustrated in Fig. 1(b), the edge molecules were to splinter nearby vertex molecules to construct longer and longer molecules representing paths in the original graph. If there exists a Hamiltonian path called for in the problem specification, one representative molecule would thus be created by the chemistry on its way to equilibrium. Using more of the same biotechnology he could then determine, as illustrated in Fig. 1(c), the presence or absence of the molecule in the final test tube and respond accordingly to the original problem.

Various factors have made molecular computing possible. Critical among them is the possibility of synthesizing biomolecules at low costs and manipulating them with relative ease despite their nanometric size. These complex molecules are composed of basic blocks called nucleotides, nucleic acid bases A (*adenine*), G (*guanine*), C (*cytosine*), T (*thymine*) (or U (*uracil*) in RNA), that bind to form chains called oligonucleotides, or $n-$mers, according to the Watson-Crick (herein abbreviated as WC) complement condition, $\overline{\mathtt{A}} = \mathtt{T}$ and $\overline{\mathtt{C}} \equiv \mathtt{G}$, and *vice versa*. Each molecule has a polarity (sense of orientation) from a so-called $5'$-end to a $3'$-end or vice versa. Their physical implementation is therefore relatively simple compared to the demanding and costly fabrication processes used in VLSI. Key manipulations include *gel Electrophoresis* (a powerful microscope that permits us to see molecules – or rather populations thereof – with the naked eye), cleaving by restriction *enzymes* (for cut-and-paste), copying by *PCR* – Polymerase Chain Reaction, and complex combinations of these basic building blocks.

The basic methodology in molecular computing to solve computational problems consists of three basic steps: an *encoding* that maps the problem onto DNA strands, *hybridization/ligation* that performs the basic core processing and *extraction* that makes the results visible to the naked eye, as illustrated in Fig. 1. Examples are *parallel overlap assembly* to generate potential solutions to problems before filtering them according to constraints at hand. (e.g., Hamiltonicity), massively paralellel *boolean-circuit evaluation*, *whiplash PCR* to implement state transitions in molecules. Potential applications include DNA fingerprinting, DNA population screening. and DNA sequencing, A 'killer application' would suit well the nature of biomolecules (e.g., bypass digitization), beat current and perhaps even future solid-state electronics, and would establish beyond the shadow of a doubt the power of the new computational paradigm, but remains to be unearthed.

The grand challenges for molecular computing are to improve its *reliability, efficiency, and scalability*. The reliability of a protocol, i.e., a DNA computation, is the degree of confidence that a lab experiment provides a true answer to the given problem. The efficiency of the protocol refers to the intended and



(a)

(b)

(c)

Fig. 1. Steps in Adleman's Molecular Computation: (a) encoding of problem instance; (b) computing reactions; (c) extraction.

effective use of the molecules that intervene in it. The scalability of a lab experiment is the effective reproducibility of the experiment with longer molecules that can encode larger problem instances while still obtaining equally realiable results under comparable efficiency. These three are distinct but clearly interrelated problems. Biologists have not really faced these problems in their work because, in that field, the definition of success is different than in computer science. (When a biologist claims that she has cloned an organism, for example, the contention is that one experiment was successful, regardless of how many were previously not, or whether only one clone was actually produced. This kind of success rate is simply unacceptableis in computer science.) Research on these problems in molecular computing has just begun. Their difficulty is rooted in our relatively poor ability to control the physical chemistry involved in the context of information processing, despite the impressive progress in biotechnology that has made it thinkable. The computation and extraction phases therefore rely on cooperative phenomena that can only be observed as *ensemble statistical processes* involving a great number of individual molecules. Success in these areas is critical for the actual construction of a molecular computing and is likely to have a feedback effect on the notions of efficiency and scalability in biology. Success will likely require self-controlled and autonomous protocols that would eliminate human intervention, and so reveal more about the true power of molecular computing. Consequently, traditional measures of algorithmic efficiency will require re-examination for measuring molecular computing efficiency.

In summary, important events have taken place in the field of biomolecular computing in the last five years. There indeed remain enormous scientific, engineering, and technological challenges to bring this paradigm to full fruition. Nonetheless, the potential gains and advantages of computing with biomolecules are likely to make it a fascinating and intriguing competitive player in the landscape of practical computing in a not too distant future.

The full version of this abstract can be found in the survey by the authors [84]. The reference list from this survey are reproduced below for the convenience of the reader. Up-to-date information can be found in several frequently updated web pages that contain more references and useful links to molecular computing. They include:
http://www.msci.memphis.edu/˜garzonm/mcg.html,
http://seemanlab4.chem.nyu.edu/, and
http://www.wi.LeidenUniv.nl/˜jdassen/dna.html.

References

1. L. Adleman, "Molecular computation of solutions of combinatorial problems," *Science* **266**, pp. 1021–1024, 1994.
2. M. R. Garey, D. S. Johnson, *Computers and Intractability.* New York: Freeman, 1979.
3. L. Landweber, L. Kari, "The Evolution of Cellular Computing: Nature's Solution to a Computational Problem," in [55], pp. 3–13.

4. A. Cukras, D. Faulhammer, R. Lipton, L. Landweber, “Chess games: A model for RNA-based computation,” in [55], pp. 15–26.
5. D. Faulhammer, A. Cukras, R. Lipton, L. Landweber, “When the Knight Falls: On Constructing an RNA Computer,” in [56], pp. 1–7.
6. R. Birge, “Protein-based Three-Dimensional Memory,” *The American Scientist* **82**, pp. 348–355, 1994.
7. R. Birge, “Protein-based Computers,” *Scientific American*, 1995.
8. A. D. Ellington, M. P. Robertson, K. D. James, J. C. Fox, “Strategies for DNA Computing,” in [21], pp. 173–184.
9. M. P. Robertson, J. Hesselberth, J. C. Fox, A. D. Ellington, “Designing and Selecting Components for Nucleic Acid Computers,” in [56], pp. 183–188.
10. E. T. Kool, “New Molecular Strategies for Joining, Pairing, and Amplifying,” in [56], pp. 62.
11. J. D. Watson, N. H. Hopkins, J. W. Roberts, J. A. Steitz, and A. M. Weiner, 4th ed., *Molecular Biology of the Gene*. Menlo Park, CA: The Benjamin/Cummings Publishing Co., Inc, 1987.
12. F. M. Ausubel, R. Brent, R. E. Kingston, D. D. Moore, J. G. Seidman, J. A. Smith, K. Struhl, P. Wang-Iverson and S. G. Bonitz. *Current Protocols in Molecular Biology*, New York: Greene Publishing Associates and Wiley-Interscience, 1993.
13. T. Head, “Formal language theory and DNA: An analysis of the generative capacity of specific recombination behaviors,” *Bull. Math. Biology*, pp. 49–73, 1985.
14. L. Landweber, R. Lipton, R. Dorit, A. Ellington (organizers), *Dimacs Workshop on Nuclei Acid Selection and Computing*, Princeton University, March 1998. http://dimacs.rutgers.edu/Workshops/NucleicAcid/index.html, http://www.princeton.edu/~lfl/poster.html.
15. Q. Ouyang, P. D. Kaplan, S. Liu, A. Libchaber, “DNA Solution of the Maximal Clique Problem,” *Science* **278**, pp. 446–449, 1997.
16. M. Arita, A. Suyama, M. Hagiya, “A Heuristic Approach for Hamiltonian Path Problems with Molecules,” in [81], pp. 457–462, 1997.
17. R. Lipton, “Using DNA to solve NP-complete problems,” *Science* **265**, pp. 542–545, 1995. See also, “Speeding up computations via molecular biology,” [53], pp. 67–74.
18. M. Ogihara, A. Ray, “DNA-Based Self-Propagating Algorithm for Solving Bounded Fan-in Boolean Circuits,” in [57], pp. 725–730.
19. M. Amos, P. E. Dunne, A. Gibbons, “DNA Simulation of Boolean Circuits,” in [57], pp. 679–683.
20. N. Morimoto, M. Arita, A. Suyama, “Solid phase DNA solution to the Hamiltonian Path Problem,” in [54].
21. H. Rubin. D. Wood (Eds.), *Proc. of the Third DIMACS Workshop on DNA-Based Computers*, The University of Pennsylvania, 1997. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Providence, RI: American Mathematical Society, vol. **48**, 1999. 185, 186, 186, 186, 188
22. M. Hagiya, M. Arita, D. Kiga, K. Sakamoto, S. Yokohama, “Towards parallel evaluation and learning of Boolean μ-formulas with molecules,” in [81], pp. 105–115.
23. K. Sakamoto, D. Kiga, K. Komiya, H. Gouzu, S. Yokohama, S. Ikeda, H. Sugiyama, M. Hagiya, “State Transitions by Molecules,” in [57], pp. 87–99, 1998.
24. E. Winfree, “Whiplash PCR for $O(1)$ Computing,” in [55], pp. 175–188.
25. M. Garzon, Y. Gao, J. A. Rose, R. C. Murphy, D. Deaton, D. R. Franceschetti, S. E. Stevens Jr. . “In-Vitro Implementation of Finite-State Machines,” *Proc. 2nd Workshop on Implementing Automata WIA-97.* Lecture Notes in Computer Science **1436**, Berlin: Springer-Verlag, pp. 56–74, 1998.

26. M. Nelson, E. Raschke, M. McClelland, “Effect of site-specific methylation on restriction endonucleases and DNA modification methyltranferases,” *Nucleic Acids Research*, **21**:13, pp. 31–39, 1993.
27. E. Shapiro, “A Mechanical Turing Machine: Blueprint for a Biomolecular Computer,” in [56], pp. 229–230.
28. L. Landweber, R. Lipton, M. O. Rabin. “DNA^{2}DNA Computation: A potential Killer-App?,” in [21], pp. 162–172.
29. R. Deaton, R. C. Murphy, M. Garzon, D. R. Franceschetti, S. E. Stevens, Jr., “Good Encodings for DNA-based Solutions to Combinatorial Problems,” in [54], pp. 159–171, 1995.
30. D. Boneh, C. Dunworth, R. J. Lipton, J. Sgall, “Making DNA Computers Error-resistant,” in [54], pp. 163–171.
31. R. Karp, C. Kenyon, O. Waarts, “Error-resilient DNA Computation,” *Proc. 7th Annual Symposium on Discrete Algorithms* SODA, pp. 458–467, 1996.
32. E. Baum, “DNA sequences useful for computation,” in [54], pp. 122–127.
33. J. G. Wetmur, “Physical Chemistry of Nucleic Acid Hybridization,” in [21], pp. 1–23.
34. J. SantaLucia, Jr., H. T. Allawi, and P. A. Seneviratne, “Improved nearest-neighbor parameters for predicting DNA duplex stability,” *Biochemistry*, vol. **35**, pp. 3555–3562, 1996.
35. R. Deaton, M. Garzon, R. C. Murphy, J. A. Rose, D. R. Franceschetti, S. E. Stevens, Jr., “On the Reliability and Efficiency of a DNA-Based Computation,” *Physical Review Letters* **80**:2, pp. 417–420, 1998.
36. A. J. Hartemink, D. K. Gifford, “Thermodynamic Simulation of Deoxyoligonucleotide Hybridization of DNA Computation,” in [21], pp. 25–37.
37. R. Deaton, D. R. Franceschetti, M. Garzon, J. A. Rose, R. C. Murphy, S. E. Stevens, Jr., “Information Transfer through Hybridization Reactions in DNA based Computing,” in [81], pp. 463–471.
38. R. Deaton, M. Garzon, J. A. Rose, D. R. Franceschetti, S. E. Stevens, Jr.. “DNA Computing: a Review,” *Fundamenta Informaticae* **35**, pp. 231–245, 1998.
39. R. Deaton, R. Murphy, J. Rose, M. Garzon, D. Franceschetti, S. E. Stevens Jr. “A DNA based Implementation of an Evolutionary Search for Good Encodings for DNA Computation,” *Proc. IEEE Conference on Evolutionary Computation*, Indiana, 267–271, 1997.
40. J. Chen, E. Antipov, B. Lemieux, W. Cedeño, D. H. Wood, “A DNA Implementation of the Max 1s Problem,” in [82], 1835-1841.
41. R. Deaton, M. Garzon, R. C. Murphy, J. A. Rose, D. R. Franceschetti, S. E. Stevens, Jr. “Genetic Search of Reliable Encodings for DNA-based Computation,” Late Breaking papers – *First Annual Genetic Programming Conference*, Stanford University, pp. 9–15, 1996.
42. J. Khodor, D. K. Gifford, A. Hartemink, “Design and Implementation of Computational Systems Based on Programmed mutagenesis,” in [55], 101–107; pp. 287–297, 1998.
43. M. Garzon, R. Deaton, J. A. Rose, D. R. Franceschetti, “Soft Molecular Computing,” in [56], pp. 89–98.
44. A. G. Frutos, Q. Liu, A. J. Thiel, A. W. Sanner, A. E. Condon, L. M. Smith, R. M. Corn, “Demonstration of a word design strategy for DNA computing on surfaces,” *Nucleic Acids Res.* **25**:23, pp. 4748–4757, 1997.
45. A. G. Frutos, L. M. Smith, R. M. Corn, “Enzymatic Ligation Reactions of DNA Words on Surfaces for DNA Computing,” *J. Am. Chem Soc.* **120**:40, pp. 10277–10282, 1998.

46. J. A. Rose, R. Deaton, D. R. Franceschetti, M. H. Garzon, S. E. Stevens, Jr., “A Statistical Mechanical Treatment of Error in the Annealing Biostep of DNA Computation,” in [82], 1829–1834.
47. C. R. Cantor, P. R. Schimmel, *Biophysical Chemistry, Part III: The Behavior of Biological Macromolecules*, New York: Freeman, 1980.
48. A. Marathe, A. E. Condon, R. M. Corn, “On Combinatorial DNA Word Design,” in [56], pp. 75–88.
49. M. Garzon, P. Neathery, R. Deaton, R. C. Murphy, D. R. Franceschetti, S. E. Stevens, Jr., “A New Metric for DNA Computing,” in [81], pp. 472–478.
50. L. M. Adleman, “On Constructing a Molecular Computer,” in [53], pp. 1–21.
51. B. T. Zhang, S. Y. Shin, “Molecular Algorithms for Efficient and Reliable DNA Computing,” in [57], pp. 735–742.
52. S. Roweis, E. Winfree, R. Burgoyne, N. V. Chelyapov, N. F. Goodman, P. W. Rothemund, L. M. Adleman, “A Sticker Based Model for DNA Computation,” in [54], pp. 1–29.
53. R. Lipton, E. Baum (Eds.), *DNA Based Computers. Proc. of the First DIMACS Workshop on DNA-Based Computers*, Princeton University, 1995. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Providence, RI: American Mathematical Society, vol. **27**, 1996. 185, 187, 188, 188
54. L. F. Landweber, E. B. Baum (Eds.), *DNA Based Computers II*, *Proc. of the Second DIMACS Workshop on DNA-Based Computers*, Princeton University, 1996. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Providence, RI: American Mathematical Society, vol. **44**, 1999. 185, 186, 186, 186, 187, 187, 188, 188
55. H. Rubin. D. Wood (Eds.), 4th DIMACS workshop on DNA Computers, University of Pennsylvania, 1998. *Proceedings* in a special issue of *Biosystems*, in press. 184, 185, 185, 186, 187, 188, 188
56. E. Winfree, D. Gifford (Eds.), *Proc. of the Fifth International Metting on DNA Based Computers*, MIT, Boston, MA. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Providence, RI: American Mathematical Society, In press. http://psrg.lcs.mit.edu/dna5/. 185, 185, 185, 186, 186, 187, 187, 188
57. J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, R. L. Riolo (Eds.), *Proc. 3rd Annual Genetic Programming Conference*, San Mateo, CA: Morgan Kaufmann, 1998. 185, 185, 185, 187, 188
58. L. M. Smith, R. M. Corn, A. E. Condon, M. G. Lagally, A. G. Frutos, Q. Liu, A. J. Thiel. “A Surface-Based Approach to DNA Computation,” *J. Comput. Biology* **5**:2, pp. 255–267, 1998.
59. M. Conrad, “On Design Principles for a Molecular Computer,” *Comm. of the Ass. Comp. Mach. CACM*, **28**:5 1985, pp. 464–480, 1985.
60. M. Conrad, “Molecular and Evolutionary Computation: the Tug of War between Context-Freedom and Context-Sensitivity,” in [55], pp. 117–129, 1998.
61. J. Chen, E. Antipov, B. Lemieux, W. Cedeño, D. H. Wood, “*In vitro* Selection for a Max 1s DNA Genetic Algorithm,” in [56], pp. 23–37.
62. R. Deaton, M. Garzon, J. A. Rose, “A DNA Based Artificial Immune System for Self-NonSelf Discrimination,” *Proc. of the IEEE Int. Conference on Systems, Man and Cybernetics*, Orlando. Piscataway, NJ: IEEE Press, pp. 369–374, 1997.
63. L. Landweber, E. Winfree, R. Lipton, S. Freeland (organizers), *Workshop on Evolution as Computation*, Princeton University, January 1999. http://dimacs.rutgers.edu/Workshops/Evolution/.
64. E. Winfree, “Universal computational via self-assembly of DNA: some theory and Experiments,” in [54], pp. 191–213.

65. E. Winfree, “Simulations of computing by self-assembly,” in [55], pp. 213–240.
66. E. Winfree. F. Liu, L. A. Wenzler, N. C. Seeman, “Design and Self-Assembly of Two-Dimensional DNA Crystals,” *Nature* **394** 1998, pp. 539–544, 1998.
67. N. Jonoska, S. A. Karl, “Ligation Experiments in DNA Computations,” *Proceedings of 1997 IEEE International Conference on Evolutionary Computation* (ICEC’97), April 13–16, pp. 261–265, 1997.
68. M. Hagiya, “Towards Autonomous Molecular Computers,” in [57], pp. 691–699, 1998.
69. M. H. Garzon, R. J. Deaton, Ken Barnes 1999. “On Self-Assembling Graphs in Vitro,” in [82], 1805–1809.
70. C. H. Bennet, “The Thermodynamics of Computation – a Review,” *Int. Journal of Theoretical Physics* **21**, pp. 905–940, 1982.
71. P. Wilhem, K. Rothemund, “A DNA and restriction enzyme implementation of Turing Machines,” in: [54] pp. 75–119, 1996.
72. W. Smith, “DNA Computers in vitro and vivo,” in: [53] pp. 121–185, 1996.
73. E. Winfree, “On the computational power of DNA annealing and ligation,” in [53], pp. 199–215, 1995.
74. G. Paun (Ed.), *Computing with Biomolecules: Theory and Experiments*. Singapore: Springer-Verlag, 1998.
75. M. Amos, A. Gibbons, P. Dunne. “The Complexity and Viability of DNA Computations,” *Proc. Biocomputing and Computation* (BCEC97), Lundh, Olsson and Narayanan (Eds.), Singapore: World Scientific, 1997.
76. S. A. Kurtz, S. R. Mahaney, J. S. Royer, and J. Simon, “Active transport in biological computing,” in [54], pp. 111–122.
77. M. Garzon, N. Jonoska, S. Karl. “The Bounded Complexity of DNA Computing,” in [55], in press.
78. J. Khodor, D. Gifford, “The Efficency of the Sequence-Specific Separation of DNA Mixtures for Biological Computation,” in [21], pp. 25–37.
79. A. J. Hartemink, T. Mikkelsen, D. K. Gifford, “Simulating Biological reactions: A Modular Approach,” in [56], pp. 109–119.
80. D. H. Wood 1998, *Basic DNA Computing*, manuscript.
81. J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, R. L. Riolo, (Eds.), *Proc. 2^{nd} Annual Genetic Programming Conference*, San Mateo, CA: Morgan Kaufmann, 1997. 185, 185, 186, 187
82. W. Bahnzhaf, A. E. Eiben, M. H. Garzon, D. E. Goldberg, V. Hanovar, M. Jakiela, J. R. Koza 1999. *Proc. of the Genetic and Evolutionary Computation Conference GECCO-99*, Orlando Florida. San Mateo, CA: Morgan Kaufmann. 186, 187, 188
83. M. Chee, R. Yang, E. Hubbell, A. Berno, X. C. Huang, D. Stern, J. Winkler, D. J. Lockhart, M. S. Morris, and S. P. A. Fodor, “Acessing genetic information with high-density DNA arrays,” *Science*, vol. **274**, pp. 610–614, 1996.
84. M. Garzon, R. Deaton, “Biomolecular Computing and Programming,” *IEEE Trans. Evol. Comput* **3**:3, 1999, in press. 184

Software Change and Evolution*

Václav Rajlich

Department of Computer Science
Wayne State University
Detroit, MI 48202, USA
rajlich@cs.wayne.edu

Abstract. Changeability (also called evolvability) is an essential property of software. Software change is the foundation for both new software development and legacy software maintenance, therefore a better understanding of software change is an important software engineering issue. This paper covers selected topics related to software change, including minicycle of change, partitioned annotations, and change propagation, and gives a brief overview of the field.

1 Introduction

A brief look at computer science textbooks and journals reveals that most of the computer science research is oriented towards the development of new software. However over the years a huge amount of software has been accumulated and is in daily use. In [9], the size of the software installed in the U.S.A. is estimated to be 36,000,000 applications of a total size of 1,700,000,000 function points, equivalent to approximately 170,000,000,000 lines of code. This software has become an integral part of the economy and plays an important role in work productivity. It would be a great omission if the software research ignored this fact and concentrated only on the development of new software. In order to account for all software, both new and old, researchers should focus on the essential properties of software.

The list of essential properties of software appeared in [3], and includes invisibility, complexity, interoperability, and changeability (also called evolvability). The experience of every software developer or maintainer confirms that software changes continually, both during development and during maintenance.

Because of the importance of change, it is a significant research goal to make software changes easy, safe, and inexpensive. One of the solutions is to anticipate changes and structure the software in such a way that the changes will be localized inside software components [12]. When a change is localized within a component, it is easier, safer, and less expensive. However, more recent research indicates that not all changes in software can be anticipated. In the case study of [6] it was reported that approximately 70 % of the requirements were predicted

* This work was partially supported by a grant from Ford Motor Co. and NSF grant CCR-9803876.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 189–202, 1999.

in advance and the remaining requirements were discovered during development. Hence the software was already undergoing changes during development, because the requirements were not accurately predicted. We believe that massive changes triggered by company mergers, introduction of Euro, emergence of graphical user interfaces and Internet, are examples of changes that could not be predicted even a few years ago. Moreover the current trend towards increasing complexity and heterogeneity of applications will make accurate prediction of changes even harder. It is likely that current applications will be exposed to many unanticipated changes during their lifetime. Therefore the support for unanticipated changes is an important research goal.

The paper is organized in the following way: Section 2 presents a minicycle of change. Section 3 presents a technique for program comprehension and redocumentation. Section 4 explains a formalism for change propagation. Section 5 gives a brief overview of the other work in program change, related to the topics of this paper. Section 6 contains conclusions and a discussion of future research.

2 Minicycle of Change

In this paper, we deal with selected aspects of software change. In order to place these aspects in the proper context, let us first introduce the so-called software minicycle, around which this paper is organized.

Software change is a process consisting of several phases:

– Request for change
– Planning phase:
  • Program comprehension
  • Change impact analysis
– Change implementation:
  • Restructuring for change
  • Change propagation
– Verification
– Redocumentation

A request for change is the specification of the change. It may be a bug report, requesting correction of a fault in software, or it may be a request for the introduction of a new functionality into the software. It may originate either from a user or a developer of software. If the change request is accepted, the change enters the planning phase.

In the planning phase, the existing software must be comprehended first, before the change can be implemented. Depending on several factors, the process of program comprehension may be easy or difficult, see Section 3.

The planning phase also contains change impact analysis, where the programmers assess the extent and the difficulty of the change. Several techniques and case studies of change impact analysis were published in the literature, see Section 5.

If the planning phase determines that the change in software is feasible, the next phase is the implementation of the change. During the change implementation, the software may be first restructured to make the change easier. The change propagation is described in Section 4.

After the change was implemented, its correctness is verified. After the verification, the new functionality and the new structure of the software is redocumented in the redocumentation phase.

It is understood that in the actual change process, the phases may overlap, be repeated, or additional phases may appear. For example if verification discovers problems, the minicycle may reenter one of the earlier phases. The actual phases of the minicycle are presented in the literature in several different forms [24], but they all capture the same basic idea. This paper avoids the complexities of the actual minicycle and presents this simplified version as a context for the techniques described in the following sections.

3 Partitioned Annotations

The first task of the programmer is to understand the program. Program understanding (or comprehension) is a subject of extensive research, see for example [13]. One of the research topics is strategies used in program comprehension.

Two specific strategies occupy a prominent role: Top-down and bottom-up [4]. In the top-down strategy, the programmer approaches the task of program comprehension by making hypotheses about the program. The programmer first accepts the fundamental hypotheses and then refines them recursively by subsidiary hypotheses. The hypotheses are verified by a search for evidence (called "beacons") in the code. If the evidence support the hypotheses, they are accepted as true. If the evidence contradicts the hypotheses, they are rejected and new ones must be formed.

Bottom-up strategy is based on chunking, which consists of two steps: aggregation and abstraction. During aggregation, the programmer scans the code and groups together constructs of the program. During the step of abstraction, the programmer recognizes these aggregates as implementations of familiar concepts. For example, several lines of code may be an implementation of a stack or an implementation of a sorting algorithm, etc. The chunks can contain other chunks as parts, and the chunking process recursively finds larger and larger chunks until the whole program is understood. The programmers usually employ a combination strategy with elements of both the top-down and the bottom-up process [22].

A part of program comprehension strategy is the fact that the programmer does not need to comprehend everything about the program in order to be able to make a change [10]. Usually it is sufficient to comprehend the relevant parts or aspects of the program. The information needs vary from task to task, and while there is a need to understand some parts of the program in great detail, other parts can be understood only roughly and still other parts scarcely at all.

Program comprehension is an expensive part of software change, absorbing on average more than half of all maintenance costs [7]. Yet with current practice, valuable knowledge gained during the change is not recorded and is in a danger of being lost when the change is completed and the programmer turns his/her attention to the next task. Partitioned annotations of software (PAS) [16] are a tool that supports the process of program comprehension by providing a notebook where the programmer can record the comprehension of the program.

PAS are structured as a matrix where one coordinate is the components of the program and the other coordinate is the partitions. The partitions are selected based on the needs of the project, and each partition contains a description of the software components from a specific point of view. For example, there can be a domain partition that describes the domain concepts that the particular component implements. Other partitions may describe the representation of concepts by data structures, archive the history of the component, document the quality of the testing, record unconfirmed hypotheses of a specific programmer about the component, etc. The selection of partitions is based on the needs of the project.

PAS allow the programmer to read only those partitions that are related to his/her specific information needs. If a component needs to be understood in detail, the programmer is likely to read all partitions for the component. If on the other hand the component needs to be understood only partially, the programmer can read only the partitions relevant to his/her task. There is no need to limit the number of partitions and amount of information stored, because unneeded or mistrusted partitions will simply not be used.

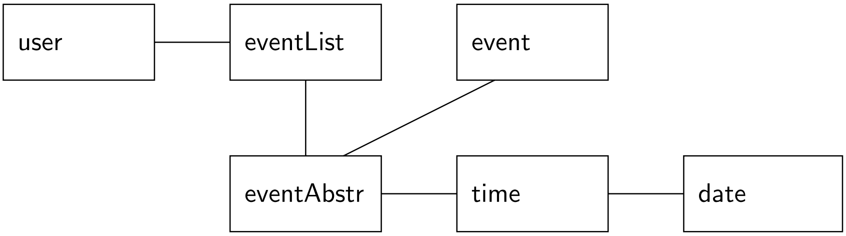


Fig. 1. Classes and interactions of Calendar program

As a small example, consider a calendar program that maintains appointments for a single user, and allows the user to enter, delete, and search appointments. The components and interactions of the program are in Figure 1, and the partitioned annotations are in Tables 1 and 2. The components of the program are classes and the partitions are the domain, representation (i.e. data structures of the class and their meaning), and interactions of the class with other classes.

For large system, PAS are implemented as hypertext [19]. In that case, standard Internet browsers like Netscape or Explorer can be used to browse through the partitions. In order to guarantee the uniformity of the annotations, a PAS generator called HMS was implemented [19] that parses the code and generates hypertext skeletons for the annotations of the program. The skeletons are then filled in by hand with the actual information.

A case study of the use of PAS in the maintenance of one million lines of C++ code was reported in [19]. In that project, PAS were used for incremental program redocumentation in the following process: Whenever a maintenance programmer completes a change, he/she produces a "release note" that summarizes the knowledge of the program gained during the change. A specially trained software documenter uses this release note and records the new knowledge in the appropriate partitions of the PAS. Over the period of 15 months, approximately 40 % of the code was redocumented and the documentation is concentrated in the most frequently maintained parts of the code. The extra cost involved in this redocumentation represents approximately a 15 % overhead, based on the time sheets produced for the customer.

Table 1. Partitioned annotation of Calendar program, part 1

user
Domain:
Class "user" implements user interface, supports insertion, deletion, and search for appointments
Representation:
N is user's name
C is instance of calendar
Interactions:
Class "user" depends on "eventList"

eventList
Domain:
Class "eventList" maintains the list of appointments, supporting addition, deletion and search
Representation:
L is list of pointers to "eventAbstr" and contains a list of appointments
Interactions:
Class "eventList" is a part of Composite pattern [8], inherits from "eventAbstr" and is used by class "user"

eventAbstr
Domain:
Class "eventAbstr" contains abstract appointment, consisting of beginning, end, and title
Representation:
Beg is beginning of appointment
End is end of appointment
Title is title of appointment
Interactions:
Class "eventAbstr" is a part of Composite pattern

For this extra expense, the following benefits were acquired: Any programmer can maintain the redocumented code without incurring the extra cost of new comprehension. This simplifies the scheduling of programmers for maintenance tasks, because the first available programmer can be assigned to do the next maintenance task. Also when the programmers leave the project, their knowledge of the code is preserved in the form of PAS annotations. The management and customers of the project consider these benefits to be worth of the extra expense.

Table 2. Partitioned annotation of Calendar program, part 2

event
Domain:
Class “event” implements a simple appointment
Representation:
Interactions:
Class “event” inherits from “eventAbstr”, part of Composite pattern

time
Domain:
Class “time” implements date, hour, and minute
Representation:
Ho contains the hour
Mi contains the minute
Interactions:
Class “time” inherits from “date”

date
Domain:
Class “date” implements date in the form mm/dd/yyyy
Representation:
Mm is 2-digit month, dd is 2-digit day, yyyy is 4-digit year
Interactions:
Class “date” is the base class for class “time”

4 Change Propagation

After the preliminary phases establish feasibility of a change, the change is implemented. The change implementation consists of several steps, each visiting one specific software component. If the visited component is modified, it may no longer fit with the other components because it may no longer properly interact with them. In that case secondary changes must be made in neighboring components, which may trigger additional changes, etc. This process is called change propagation. Although each change starts and ends with consistent software, during the change propagation the software is often inconsistent.

This process is modeled by the graph rewriting of [14,11]. The basic notion is an evolving interoperation graph (eig), defined in the following way: Let C be a set of components of the program. An *interoperation* or *interaction* between two components $b, c \in C, b \neq c$ is formally represented as an unordered couple $\{b, c\}$. *Interoperation graph* G is a set of interoperations. Examples of interoperations are function calls, data flows, use of shared resources, etc. The changes propagate through interoperations from one component to the next.

For interoperation graph G, interoperation $\{b, c\} \in G$ is *marked* if there exists an ordered couple $\langle b, c\rangle$ called *mark.* Existence of a mark intuitively means that component b was changed or inspected in the past, and component c will be changed or inspected in the future. Marked interaction can be inconsistent, i.e. there could be a conflict in the interaction of the two components. An example of such a conflict is a different number of arguments in a function call than in the function declaration. If $\langle b, c\rangle \in E$, then c is called a *marked* component. An *evolving interoperation graph* (eig) is a set E of interoperations and marks such that $\langle b, c\rangle \in E$ implies $\{b, c\} \in E$. Eig E is unmarked if there are no marks in E and marked if there are marks in E.

For eig E, define the following sets:

$$\begin{aligned} G(b) &= \{\{b,c\} | \{b,c\} \in E\} && \text{(interoperations of } b\text{)} \\ \underline{M}(b) &= \{\langle c,b\rangle | \langle c,b\rangle \in E\} && \text{(incoming marks to } b\text{)} \\ M(b) &= \{\langle b,c\rangle | \langle b,c\rangle \in E\} && \text{(outgoing marks from } b\text{)} \\ E(b) &= G(b) \cup \underline{M}(b) \cup M(b) && \text{(evolving neighborhood of } b\text{)} \end{aligned}$$

A *visit* to a component is the replacement of that component and its neighborhood by an updated one. Formally, let b and b' be a component before and after the visit, $E(b)$ and $E'(b')$ be the evolving neighborhood before and after the visit, and E and E' be an eig before and after the visit, respectively. Then a *visit* is a couple of eigs $\langle E, E'\rangle$ such that $E' = (E - E(b)) \cup E'(b')$. A scenario is a sequence of evolving dependency graphs $E_1, E_2, \ldots E_n$ that starts and ends with unmarked graphs, i.e. both E_1 and E_n are unmarked. If there are no backtracks, then for each step i, $\langle E_i, E_{i+1}\rangle$ is a visit. If there are backtracks in the scenario, then either $\langle E_i, E_{i+1}\rangle$ is a visit, or for some k, $0 < k < i$, $E_{i+1} = E_k$.

The *strategy* of a specific scenario is a set of constraints imposed on the visits. This paper considers finality, accuracy, and strictness. If visits are *final,* then we assume that after visiting component b and then its neighbor component c, component b does not need another immediate visit. A non-final strategy means that there may be an immediate need to revisit b again. Formally, this is expressed in the following way:

- For a non-final change, $\underline{M}'(b') = \emptyset$ and $M'(b') \subseteq \{\langle b', c\rangle | \{b', c\} \in E'(b')\}$.
- For a final change, $\underline{M}'(b') = \emptyset$ and $M'(b') \subseteq \{\langle b', c\rangle | \{b', c\} \in E'(b')\} - \{\langle b', d\rangle | \langle d, b\rangle \in \underline{M}(b)\}$.

The accuracy is dependent on the depth of program analysis. This paper assumes that the program analysis is limited to a static analysis, that extracts only the components and the existence or nonexistence of interactions among them, without any additional information. In this situation, we will distinguish between two possibilities: Either the change propagates through the component to all neighbors, or none at all. The distinction cannot be done automatically by the analysis tool, but must be made by the programmer, based on his/her knowledge of semantics. In the first case, for non-final change $M'(b') = \{\langle b', c\rangle | \{b', c\} \in G'(b')\}$ and for final change, $M'(b') = \{\langle b', c\rangle | \{b', c\} \in G'(b')\} - \{\langle b', d\rangle | \langle d, b\rangle \in \underline{M}(b)\}$. In the second case, $M'(b') = \emptyset$. This will be called *coarse* accuracy.

Another constraint is *strictness.* Strict strategies visit only marked components, and lenient strategies can visit any component at any time.

A more complete exposition of these concepts can be found in [17,20]. The concepts are illustrated by the following example.

Example

As an example of a change, consider an evolution of a calendar program of Figure 1. The components of the program are classes and the interactions between the components are relations of inheritance and agregation among them. Formally, the program is described by eig

$$P_1 = \{\{\text{user}, \text{eventList}\}, \{\text{eventList}, \text{eventAbstr}\}, \{\text{eventAbstr}, \text{event}\}, \{\text{eventAbstr}, \text{time}\}, \{\text{time}, \text{date}\}\}$$

The functionality of the components and the interactions are explained in the annotations in Tables 1 and 2. In this program, we are going to change the insertion of a new appointment. If the user will want to enter a new appointment that involves a weekend, he/she will be warned and prompted to confirm or cancel such an appointment. This new functionality will be added to the program in a change propagation scenario that uses a coarse, strict, and final strategy.

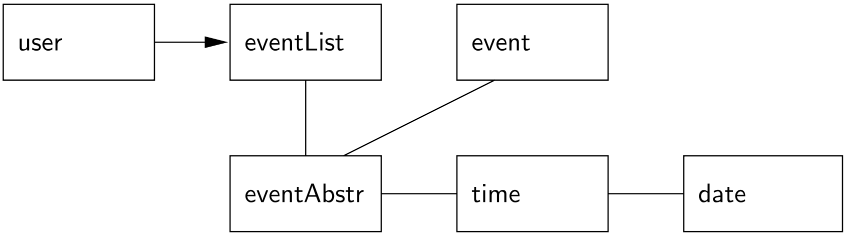


Fig. 2. Calendar program after change in class "user"

The change starts in the class "user" by a modification of the method that inserts new appointments. The new method prompts the user to confirm or cancel the appointment when the dates of the appointment conflict with a weekend. The information about weekend comes from the class "eventList" and the old version of the class does not provide it. Hence after the change in class "user", the program is inconsistent and it is represented by eig in Figure 2, where the arrow represents the mark (or inconsistency). The eig of Figure 2 is formally described as $P_2 = P_1 \cup \{\langle \text{user}, \text{eventList}\rangle\}$.

The next class to be changed is the class "eventList". The method to be changed is the method that includes a new appointment into the list of appointments. The old version of that method returns a value that indicates whether the new appointment conflicts with any existing appointments. In the new version, the range of returned values has to be extended to indicate whether the new appointment conflicts with a weekend. The new version of the method invokes another method that will identify whether an event conflicts with a weekend. That method, like all time information, is inherited from "eventAbstr" and hence the change propagates further. The new eig after the visit is in Figure 3. Formally it is described as $P_3 = P_1 \cup \{\langle \text{eventList}, \text{eventAbstr}\rangle\}$.

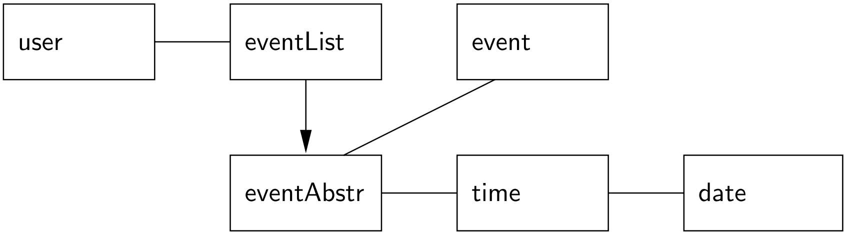


Fig. 3. Calendar program after change in "eventList"

The class "eventAbstr" is an aggregate of two instances of "time", one for the beginning of the appointment and the other one for the end. The method that determines whether the new appointment conflicts with a weekend has to be added to "eventAbstr". In accordance with the final and coarse strategy of propagation, both "event" and "time" have to be marked, see eig in Figure 4. Formally it is described as $P_4 = P_1 \cup \{\langle \text{eventAbstr}, \text{event}\rangle, \langle \text{eventAbstr}, \text{time}\rangle\}$.

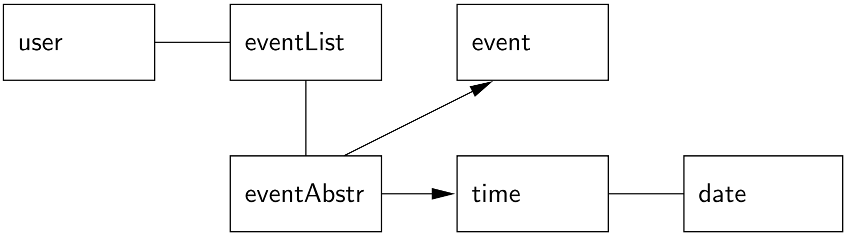


Fig. 4. Calendar program after visit to "eventAbstr"

The class “event” inherits all time information from “eventAbstr” and therefore the visit confirms that “event” does not need any change. The visit to the class “time” reveals that all information related to dates is inherited from the class “date”, so no change is required in that class. In spite of that, the change propagates further to the class “date”, see new eig in Figure 5. Formally, $P_5 = P_1 \cup \{\langle \text{time}, \text{date} \rangle\}$.

The class “date” is the last visit of the change propagation scenario. In this visit, we provide a new method that identifies for a given date the nearest following weekend date. After the method has been added, the program is consistent again, and now supports the new feature that identifies appointments conflicting with weekends.

This completes the example of change propagation. Please note that all of the classes of the program had to be visited in the scenario. All of them except two had to be changed. One class (“time”) did not need any change, yet the change propagated through it to the next class (“date”).

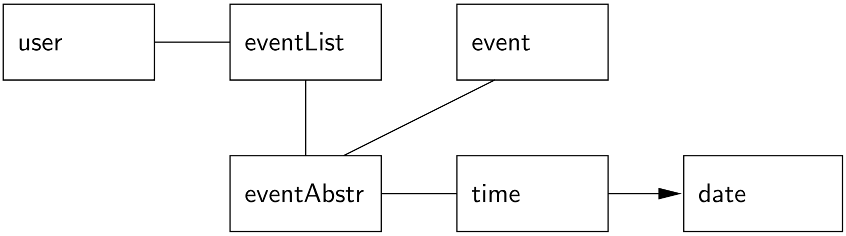


Fig. 5. Calendar program after visit to “time”

5 Other Work

In the previous sections, we described techniques dealing with program comprehension and change propagation. This section briefly reviews other work related to these topics.

5.1 Browsers

Browsers are software tools that extract information about components and their interactions and store this information in a database. Examples of interactions are relationships between the definitions and the use of variables, procedures, types, etc. The user queries the database to navigate through the software, following the dependencies among components. In order to enhance understanding, the results of the queries are often displayed graphically. Since change propagates through the dependencies from one component into another, browsers support change propagation scenarios. Examples of browsers are [5,15].

It should be noted that the dependencies among software components are used in configuration management [21] for a simple form of change propagation: If a specific component changes, all components recursively depending on it must change. Compared to the scenarios of Section 4, this is a highly special case of change propagation, where the change propagates through the dependencies only in one direction, e.g. from the used to the using components. Note that in the example in Section 4, the change propagates in exactly the opposite direction. Another difference is the fact that during the change propagation scenarios, the interaction can be inconsistent while configuration management assumes a consistency. Also the configuration management marks all components that are recursively dependent on a changed component, while change propagation scenarios allow the change propagation to stop, without requiring a change in all recursively dependent components.

5.2 Change Impact Analysis

Change impact analysis is a process by which the programmers identify the components that will be impacted by the change. The purpose of a change impact analysis is to assess the cost and difficulty of a change before it is actually undertaken. A representative selection of articles on change impact analysis appears in [2].

Change impact analysis includes concept location, by which a programmer locates a specific concept in the code (for example the conversion of a date to a day). Change requests are often formulated in terms of domain concepts, therefore the recognition and location of concepts in the code is a key part of change impact analysis. In [1], the authors discuss a scenario of concept recognition based on reading a static program. Their strategy utilizes various clues, including identifier names and clusters of the function calls that reveal the location of the concept in the code.

In [23], the features in the code are located by dynamic analysis. For that, the program is instrumented in such a way that the statements executed in a test are marked, and statements that were not executed are left unmarked. Then the program is executed two times, once with the feature being executed and once without the feature. The feature must be located within the statements that were executed the first time but not second time.

5.3 Evolvable Architectures

Since software evolution is inevitable, it is very important to make program architectures evolvable. Since program comprehension and change propagation are the most expensive parts of changes, the programs will be more evolvable if they are more comprehensible and if the change propagation is short.

The programs can be made more comprehensible by having important domain concepts localized inside program components. A large part of program comprehension effort is geared towards establishing the traceability between domain concepts and software components, and an architecture that has a simple traceability is easier to comprehend. The change requests are often framed in terms of domain concepts, as we saw in the example of the Calendar where the domain notions "appointment" and "weekend" were used. If the changing concept is fully localized within one component, the change propagation will be short. This is the thrust of the recommendation of [12]. However it should be noted that the concepts are often intertwined in various ways, and while some concepts are localized, others are inevitably delocalized.

The studies of evolvable software architectures sometimes provide surprising answers. In a case study of [18], we studied the evolvability of a software repository, i.e. a program that is data oriented and the user manipulates the data through use cases, selectable from a menu. A typical evolution of such system is the addition of new use cases, which will allow additional data manipulations. We found that in this case, a program with use cases implemented as C functions and data implemented as SQL relational tables were more evolvable, than a program with the same functionality and implemented as a C++ object oriented program. The reason for this is in the fact that the use cases are the most important concepts in this case, and in object oriented programs, they are divided into small functions and delocalized into separate classes. Hence in this particular case, the functional/relational implementation has a better concept locality than the corresponding object oriented program.

6 Conclusions and Future Work

In this paper, we gave an overview of selected topics in software change and evolution. The study of this topic gives insight into how software changes are made and how they can be improved.

A topic for future research is the implementation of a tool that supports change propagation. Such a tool must be able to analyze the program and store the results of the analysis in a database, similarly as browsers. However it has to have several additional capabilities. It should be able to analyze inconsistent programs, i.e. programs where interactions between the components are inconsistent, as is often the case during the change propagation. Also it should be able to support different strategies of change propagation, by keeping the marks and automatically updating them based on the selected strategy.

The strategies and scenarios of change propagation are also topics for future research. In this paper, we discussed the finality, strictness, and accuracy of scenarios, but there are additional constraining factors, some of them dependent on a deeper analysis of the program. This deeper analysis should take into account component and interaction semantics and improve the accuracy of the marking after a change. That would relieve the programmer from the necessity to visit components that do not need change and do not propagate change.

The situation where a component does not need a change but propagates it to its neighbors is intuitively a very likely place where the programmer may make an error. This observation is supported by observations from practice, where one of the most common sources of the errors is forgotten update. Again a deeper semantic analysis of the programs may warn about some of these situations.

Research in software change and evolution deals with an essential property of software. In this paper, we presented two specific solutions to partial problems: partitioned annotations of software for program comprehension, and evolving interoperation graphs for change propagation. The research topics in this field are interesting and applicable to programming practice. That should be attract researchers to this field.

References

1. T. J. Biggerstaff, B. G. Mitbander, D. E. Webster, Program Understanding and the Concept Assignment Problem, Comunications of ACM, May 1994, 72–78. 199
2. S. A. Bohner, R. S. Arnold, ed., Software Change Impact Analysis, IEEE Computer Soc. Press, Los Alamitos, CA, 1996. 199
3. F. Brooks, No Silver Bullet, IEEE Computer, April 1987, 10–19. 189
4. R. Brooks, Towards a Theory of the Cognitive Processes in Computer Programming, Int. J. Man-Machine Studies, Vol. 9, 1977, 737–751. 191
5. Y. F. Chen, M. Y. Nishimoto, C. V. Ramamoorthy, The C Information Abstractor System IEEE Transactions on Software Engineering Vol. 16, 1990, 325–334. 198
6. M. A. Cusumano, R. W. Selby, How Microsoft Builds Software, Communications of ACM, June 1997, 53–61. 189
7. R. K. Fjeldstad, W. T. Hamlen, Application Program Maintenance Study: Report to Our Respondents, in G. Parikh, N. Zvegintzov, eds., Tutorial on Software Maintenance, IEEE Computer Society Press, Los Alamitos, CA, 1982, 13–30. 191
8. E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Addison Wesley, Reading, MA, 1995. 193
9. C. Jones, The Year 2000 Software Problem, Addison Wesley, Reading, MA., 1998. 189
10. A. Lakhotia, Understanding Someone Else's Code: An Analysis of Experience, J. Systems and Software, 1993, 269–275. 191
11. D. Le Metayer, Describing Software Architecture Styles Using Graph Grammars, IEEE Trans. On Software Engineering, 1998, 521–533. 194
12. D.L. Parnas, On the criteria to be used in decomposing systems into modules, Communications of the ACM, Dec. 1972, 1053–1058. 189, 200
13. Proceedings of the 7th IEEE International Workshop on Program Comprehension, IEEE Computer Society Press, Los Alamitos, CA, 1999. 191
14. V. Rajlich, Theory of Data Structures by Relational and Graph Grammars, In Automata, Languages, and Programming, Lecture Notes in Computer Science 52, Springer Verlag, 1977, 391–511. 194
15. V. Rajlich, N. Damaskinos, P. Linos, W. Khorshid, VIFOR: A Tool for Software Maintenance, Software – Practice and Experience, Vol. 20, No. 1, Jan. 1990, 67–77. 198
16. V. Rajlich, S. R. Adnapally, VIFOR 2: A Tool for Browsing and Documentation, Proc. IEEE International Conference on Software Maintenance, IEEE Computer Society Press, Los Alamitos, CA, 1996, 296–299. 192
17. V. Rajlich, A Model of Change Propagation based on Graph Rewriting, Proc. Of IEEE International Conf. On Software Maintenace, IEEE Computer Society Press, Los Alamitos, CA, 1997, 84–91. 196

18. V. Rajlich, S. Ragunathan, A Case Study of Evolution in Object Oriented and Heterogenous Architectures, The Journal of Systems and Software 43, 85–91, 1998. 200
19. V. Rajlich, S. Varadajan, Using the Web for Software Annotations, International Journal of Software Engineering and Knowledge Engineering 9, 1999, 55–72. 192, 192, 193
20. V. Rajlich, Modeling Software Evolution by Evolving Interoperation Graphs, to be published in Annals of Software Engineering, Vol. 9, 2000. 196
21. W. Tichy, Configuration Management, John Wiley and Sons, N.Y., 1994. 199
22. A. von Mayrhauser, A. Vans, Comprehension Processes During Large Scale Maintenance, Proc. 16th Int. Conf. on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, May, 1994, 39–48. 191
23. N. Wilde, R. Huitt, Maintenace Support for Object-Oriented Programs, Proc. Conf. on Software Maintenance, IEEE Computer Society Press, Los Alamitos, CA, 1991, 162–170. 199
24. S. S. Yau, J. S. Collofello, T. MacGregor, Ripple Effect Analysis of Software Maintenance, Proc. Compsac, IEEE Computer Society Press, IEEE Computer Society Press, Los Alamitos, CA, 1978, 60–65. 191

Distributed Simulation with Cellular Automata: Architecture and Applications

P. M. A. Sloot, J. A. Kaandorp, A. G. Hoekstra, and B. J. Overeinder

Faculty of Sciences, University of Amsterdam
Section Computational Science
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{sloot,jaapk,alfons,bjo}@wins.uva.nl
www.wins.uva.nl/research/scs/

1 Introduction

Many fundamental problems from natural sciences deal with complex systems. We define a complex system as a population of unique elements with well defined microscopic attributes and interactions, showing emerging macroscopic behavior. This emergent behavior can, in general, not be predicted from the individual elements and their interactions. A typical example of emergent behavior is self-organization, e.g. Turing patterns in reaction-diffusion systems. Complex systems are often irreducible[1] and can not be solved in an analytical way. The only available option to obtain more insight into these systems is through explicit simulation. Many of these problems are intractable: in order to obtain the required macroscopic information, extensive and computationally expensive simulation is necessary. Since simulation models of complex systems require an enormous computational effort, the only feasible way is to apply massively parallel computation. A major challenge is to apply High Performance Computing in research on complex systems and, in addition, to offer a parallel computing environment that is easily accessible for applications [62,63].

Traditionally, science has studied the properties of large systems composed of basic entities that obey simple microscopic equations reflecting the fundamental laws of nature. These natural systems may be studied by computer simulations in a variety of ways. Generally, the first step in any computer simulation is to develop some continuous mathematical model that is subsequently discretized for implementation on a computer. An alternative, less widely used approach is to develop solvers that conserve the characteristic intrinsic parallel properties of the applications and that allow for optimal mapping to a massively parallel computing system. These solvers have the properties that they map the parallelism in the application via a simple transformation to the parallelism in the machine. With these transformations the necessity to express the application into complex mathematical formulations becomes obsolete.

One example is the modeling of a fluid flow. Traditionally this problem is simulated through mathematical description of the phenomenon via Navier-Stokes

[1] Irreducible problems can only be solved by direct simulation

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 203–248, 1999.

equations, and discretization of these equations into numerical constructs for algorithmic presentation on a computer. This process of simulation involves a number of approximations and abstractions to the real fluid flow problem: intrinsic properties and explicit information of the physical phenomenon is obscured. Even worse, the possible implicit parallelism of the problem becomes completely indistinct in the abstraction process. An alternative approach would be to model the microscopic properties of the fluid flow with cellular automata, where the macroscopic processes of interest can be explored through computer simulation. This approach has the advantage that the physical characteristics of the fluid flow problem remain visible in the solving method and that the method conserves the parallelism in the problem. Although this type of simulation methods is not yet completely understood and certainly not fully exploited, it is of crucial importance when massively parallel computers are concerned. We define this type of solvers as *natural solvers*. These techniques have in common that they are inspired by processes from nature [64]. Important examples of natural solvers are Genetic Algorithms (inspired by the process of natural selection), Simulated Annealing (inspired by the process of cooling heated material which converges to a state of minimal energy), Lattice Gases and the Lattice Boltzmann method (a many particle system, or cellular automaton method with a macroscopic behavior that corresponds to the hydrodynamic equations), and artificial Neural Networks. We argue that in parallel computing the class of natural solvers results in a very promising approach, since the physical characteristics of the original phenomenon remain visible in the solving method and the implicit and explicit parallelism of the problem remain conserved.

In Fig. 1 a “bird’s eye view” of the different steps of the mapping process from application to parallel machines is presented. As can be seen, an application is first transformed into a solver method. Here, detailed knowledge of the problem domain is obligatory. Next, the intrinsic parallelism in the solver is passed through the Decomposition layer that captures the parallelism and dependencies into objects and communication relationships. Finally these two classes are mapped onto a Virtual Parallel Machine model that allows for implementation on a large suite of parallel systems [52].

To be able to capture the generic aspects of parallel solvers and to express the basic properties of the natural system, we will define our own abstract solver model indicated as the Virtual Particle model. The Virtual Particle (VIP) can be defined as the basic element in the simulation model. The VIP can be defined on several levels of abstraction. For example in a simulation model of a biological system, the VIP can correspond to a certain level of organization and aggregation in the system (e.g. molecule-organelle-cell-tissue-organ-organism-population). The choice of the abstraction level is determined by a combination of the desired refining of the model and the computational requirements. In the VIP model the microscopic, temporal or spatial, rules have to be specified in such a way that they approximate the microscopic rules as observed in the actual system. In the VIP model, the VIPs may correspond to the individual particles in the natural solver, as for example in lattice gases. Alternatively,

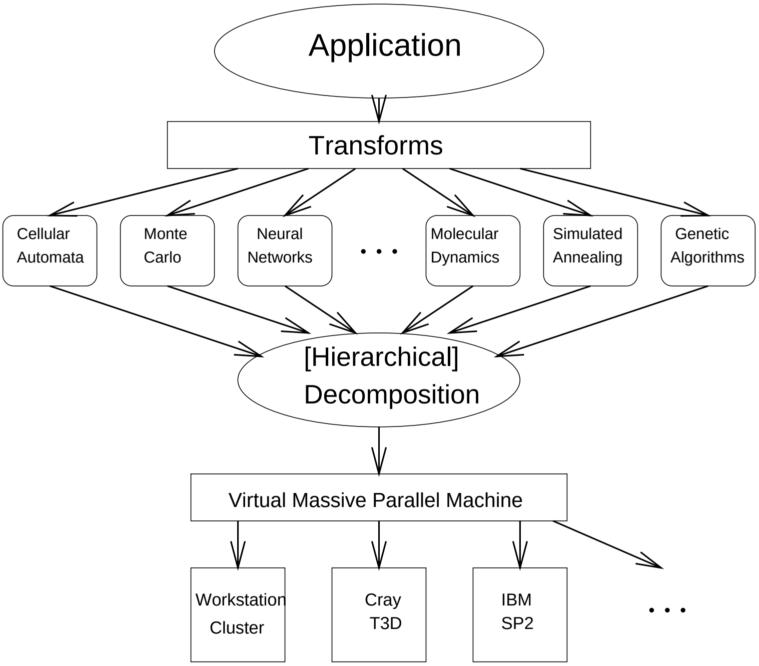


Fig. 1. Outline of a Parallel Programming Model for Dynamic Complex Systems on Massively Parallel Computers

the particles can be organized hierarchically, where VIPs can be an individual particle or clusters of VIPs. The application model is mapped onto the Virtual Parallel Machine Model (see Fig. 2), which can be another instance of a dynamic complex system consisting of a population of processors. In this case both load balancing and minimization of communication can be taken into account in a graph representation [56,59].

In this paper we will focus on cellular automata methods for modeling phenomena from natural sciences. In Section 2 the theoretical background of Cellular Automata (CA) will be briefly discussed. In Section 3 different execution models for CA will be discussed. Section 4 presents an example of a very specific CA that can be used as a model of fluid flow. In Section 5 we will demonstrate the use of two types of execution models. The first application shows how synchronous cellular automata can model growth processes in a moving fluid. The second application demonstrates an asynchronous execution scheme for a continuous-time Ising spin model.

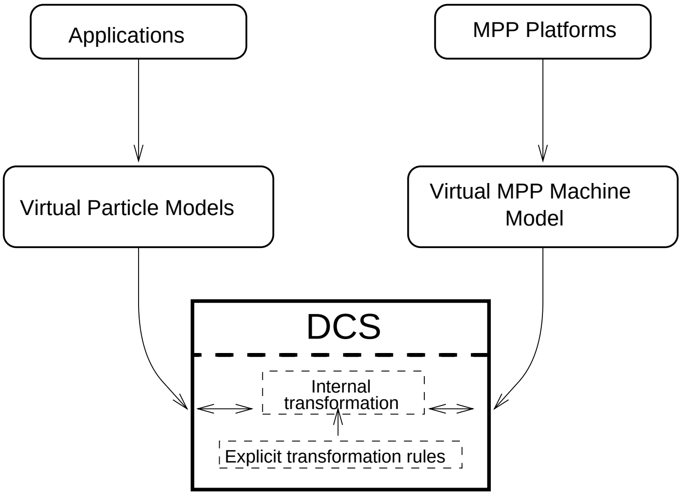


Fig. 2. Basic structure of Dynamic Complex System Paradigm: the mapping of the application model onto the machine model. The internal transformation denotes the mapping of the application graph onto the machine graph

2 Background of Cellular Automata Concepts

2.1 Introduction

Cellular Automata are discrete, decentralized, and spatially extended systems consisting of large numbers of simple identical components with local connectivity. The meaning of discrete here is, that space, time, and features of an automaton can have only a finite number of states. The rational of cellular automata is not to try to describe a complex system from a global point of view as it is described using for instance differential equations, but modeling this system starting from the elementary dynamics of its interacting parts. In other words, not to describe a complex system with complex equations, but let the complexity emerge by interaction of simple individuals following simple rules. In this way, a physical process may be naturally represented as a computational process and directly simulated on a computer. The original concept of cellular automata was introduced by von Neumann and Ulam to model biological reproduction and crystal growth respectively [66,68]. Von Neumann was interested in the connections between biology and computation. Specifically the biological phenomenon of self-reproduction modeled by automata triggered his research in this field. According to Burks [7], Stanislaw Ulam suggested the notion of cellular automata to von Neumann as a possible concept to study self-reproduction. Since then it has been applied to model a wide variety of (complex) systems, in particular physical systems containing many discrete elements with local interactions [44,73]. Cellular Automata have been used to model fluid flow, galaxy formation, biological pattern formation, avalanches, traffic jams, parallel computers, earthquakes, and many more. In these examples, simple microscopic rules display macroscopic emergent behavior. For some Cellular Automata it can be proven that they are equivalent to Universal Computers, thus in principle able to compute any given algorithm, comparable to Turings Universal Computing Machine (see for instance [5]). Furthermore Cellular Automata can provide an alternative to differential equations for the modeling of physical systems. It is this combination of

- simple local rules,
- association with universal computing automata,
- alternative to differential equations,
- models of complex systems with emergent behavior,
- and bridging the gap between microscopic rules and macroscopic observables

that has renewed interest in Cellular Automata (CA's). The locality in the rules facilitate parallel implementations based on domain decomposition, the Universal Computing behavior supports fundamental research into the intrinsics of computation in CA's, and the modeling power of CA's is of utmost importance to study a huge variety of complex systems. Although John von Neumann introduced the cellular automata theory several decades ago, only in recent years it became significant as a method for modeling and simulation of complex systems. This occurred due to the implementation of cellular automata on massively parallel computers. Based on the inherent parallelism of cellular automata, these new architectures made possible the design and development of high-performance software environments. These environments exploit the inherent parallelism of the CA model for efficient simulation of complex systems modeled by a large number of simple elements with local interactions. By means of these environments, cellular automata have been used recently to solve complex problems in many fields of science, engineering, computer science, and economy. In particular, *parallel* cellular automata models are successfully used in fluid dynamics, molecular dynamics, biology, genetics, chemistry, road traffic flow, cryptography, image processing, environmental modeling, and finance [65].

2.2 Simple 1D Cellular Automata

A CA consists of two components: a lattice of N identical finite-state machines called cells, each with an identical pattern of local connections to other cells for input and output, and a transition rule. Let Σ denote the set of states in the cell's finite state machine, and $k = |\Sigma|$ denote the number of states per cell. The state of cell i at time t is denote by s_i^t, with $s_i^t \in \Sigma$. The state of cell i together with the states of the cells to which cell i is connected is called the neighborhood n_i^t of cell i. The transition rule $\phi\left(n_i^t\right)$ gives the updated state s_i^{t+1} for each cell i as a function of n_i^t. In a CA a global clock usually provides the update signal for all the cells; cells update their states *synchronously.* In Section 3 we will discuss the consequences of *asynchronous* updates.

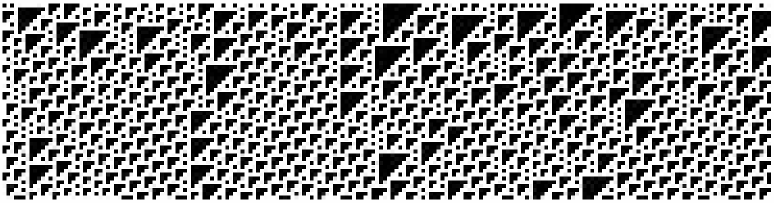


Fig. 3. The "space-time" behavior of CA 110, where black = 0 and white = 1. Space is on the horizontal axis, time flows from top to bottom

Consider for instance the following one-dimensional CA with $k = 2$ (i.e. $\Sigma = \{0, 1\}$) and a transition rule given by

$$\begin{array}{lcccccccc} \text{Neighborhood:} & 111 & 110 & 101 & 100 & 011 & 010 & 001 & 000 \\ \text{Output bit:} & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{array} \qquad (1)$$

then the transition rule number (due to Wolfram [73]) is given by "110", being the decimal representation of the output bit-string "01101110". The "space-time" behavior of CA 110 is shown in Fig. 3. Starting from a random initialization of the CA with length 250, the transition rule is iteratively applied 50 time steps. The "time" runs from top to bottom. For a one-dimensional CA the size of the neighborhood n_i^t is given by $2r+1$ with r the radius of the CA (in CA 110, $r = 1$). Wolfram studied in great detail the 256 possible one-dimensional $k = 2$, $r = 1$ CAs (so-called elementary CAs) and classified them accordingly to dynamical systems [72].

- Class 1: Fixed point behavior. Almost all initial configurations relax after a transient period to the same fixed configuration.
- Class 2: Periodic behavior. Behavior like in class one but with temporally periodic cycles of configurations included.
- Class 3: Chaotic behavior. Unpredictable space-time behavior.
- Class 4: Complex behavior. Complicated localized patterns occur, sometimes "long-lived".

For infinite Class 4 CAs it is effectively undecidable whether a particular rule operating on a particular initial seed will ultimately lead to a frozen state or not, this is the CA analog of Turings Halting problem. It is speculated that all CAs with Class 4 behavior are capable of universal computation [72]. In 1990 it was shown by Lindgren and Nordahl for the first time that a one-dimensional CA ($r = 1$, $k = 7$) exhibit universal computing behavior [38]. If Class 4 CAs are capable of universal computing, then CA 110, shown in Fig. 3, is a very good *elementary* CA candidate for universal computation [37].

2.3 Requirements for Computability in Cellular Automata

Some authors [13,35,72] suggest that when a system displays complex behavior, universal computations can be performed. Mechanically speaking a computational system requires *transmission*, *storage* and *modification* of information. So, whenever we identify those three components in some dynamical system, the system could be computationally universal. But then the question remains when does this happen? Loosely speaking we can say, using information theoretical results, that it must take place at an intermediate level of entropy: stored information lowers the entropy, but transmission of information increases the entropy level. Therefore we will briefly review entropy measures of Cellular Automata in Section 2.3.1. In a number of papers, Christopher Langton has tried to answer this question by considering Cellular Automata as a theoretical model for a physical system. The hypothesis "Computation at the Edge of Chaos" resulted from this research. Briefly it states that universal computations can take place at the border between order and chaos. This statement partly resulted from the observation that correlations can become infinite during, or at a *second order phase transition* between, for example, a solid and a liquid phase. Recall that a discontinuous change in an order parameter of the system corresponds to a first order transition. A sudden, but continuous, change corresponds to a second order transition. At such a transition, the system is in a *critical* state. Langton and Kauffman [33,35] believe that the resulting infinite correlations can be interpreted as long-term memory needed to store information. We will review these notions and recent objections to this hypothesis briefly in Section 2.3.2.

2.3.1 Information in CA In order to observe phase transitions in CA evolution, quantitative order parameters are needed. These order parameters need to distinguish between ordered and disordered states. A commonly used quantity for this purpose is the Shannon entropy [60], defined on a discrete probability distribution p_i:

$$H = -\sum_i p_i log p_i \tag{2}$$

This measure H can be associated with the degree of uncertainty about a system. In a Cellular Automata, entropy can be defined on the k^N possible subsequences of N-length blocks in a k-state CA. In a random sequence all subsequences must occur with equal probability. With probabilities p_i for the k^N possible subsequences:

$$H_N = -\sum_{i=1}^{k^N} p_i log p_i \tag{3}$$

The *spatial block entropy* [21] is now defined as:

$$h_N^{(x)} = H_{N+1} - H_N \tag{4}$$

The superscripts (x) indicate that spatial sequences are considered. From Eq. 4 follows the *spatial measure entropy*, or *entropy rate* [21]:

$$h^{(x)} = \lim_{N \to \infty} h_N^{(x)} \tag{5}$$

The measure entropy gives the average information content per site. Analogous to the spatial entropy, one can define *temporal entropy*, where blocks of $N \times T$ sites are considered:

$$h^{(t)} = \lim_{N,T \to \infty} h_{N,T}^{(t)} \tag{6}$$

Eq. 4 decreases monotonically with N, while $h_{N,T}^{(t)}$ decreases with T. The difference,

$$\delta h_N^{(x)} = h_N^{(x)} - h_{N+1}^{(x)} \tag{7}$$

is the amount of information by which a state s_{i+N} of a cell $i+N$ becomes less uncertain if the cell state s_i is known. $\delta h_N^{(x)}$ is called the N-th order mutual information in space. Intuitively one could regard mutual information as the stored information in one variable about another variable and the degree of predictability of a second variable by knowing the first.

The space-time (random) processes that occur in deterministic CA can also be studied through Kolmogorov-Sinai entropy per unit time [19]: the so-called (ε, τ)-entropy

$$h^{\text{space, time}}(\varepsilon, \tau) = \lim_{TV \to \infty} \frac{1}{TV} H(\varepsilon, \tau, T, V)\,,$$

where $H(T, V)$ denotes the entropy of the process over a simulation time T and volume V. For instance $H(T, V)$ for rule 132 $\sim V \log T$ and for rule 250 where the initial state is attracted towards spatially uniform (or periodic) configuration $H(T, V) \sim \log TV$. In all cases the entropy per unit time and unit volume $(h^{\text{space, time}}(\varepsilon, \tau))$ is vanishing.

2.3.2 Computation in Cellular Automata If we have a k-state CA with a neighborhood size r, the total number of possible transition rules is $k^{k^{2r+1}}$, which can become very large, even for a moderate number of states and/or a small neighborhood. If a structure is present in this enormous space, it should be possible to identify areas of equal complexity (i.e. the Wolfram classes) and show how these areas are connected to each other. Using this ordering one can locate those areas which support the transmission, storage and modification of information. Langton [35,36], suggested the parameter λ to structure the CA rule-space. An arbitrary state $s \in \Sigma$ is assigned the *quiescent state* s_q. Let there be n transitions to this quiescent state in an arbitrary transition rule. The remaining $k^r - n$ transitions are filled randomly by picking uniformly over the other $k - 1$ states:

$$\lambda = \frac{k^r - n}{k^r} \tag{8}$$

If $\lambda = 0.0$ then all transitions in the rule will be to the quiescent state s_q. If $\lambda = 1.0$ there will be no transitions to s_q. All states are represented equally in the rule if $\lambda = 1 - 1/k$. With the aid of the λ-parameter it should be possible to examine the assumption that *complex behavior* is located at the intermediate regime between ordered and disordered behavior. The spectrum of dynamical behavior can be explored with the so-called table-walk-through-method which increases the λ-parameter at successive time steps. At each new time step a transition table is incrementally updated using the transition table at the previous time step. Because the described method is actually a "random walk" through a coarse grained version of the CA state-space, each table-walk displays quantitatively different behavior. Several measures can be used to characterize the dynamical behavior of the CA at each new value of the λ-parameter. These measures include the numerical determination of block entropies, and both temporal and spatial mutual information statistics.

At intermediate values of λ, i.e. at the edge between ordered and disordered dynamics, several events seem to occur:

- transient lengths grow rapidly, analogously to the physical event of *critical slowing down*,
- transient lengths depend exponentially on the size of the CA,
- mutual information measures (see Eq. 7) reach their maximum values, see Fig. 4 (left), at the entropy transition, see Fig. 4 (right).

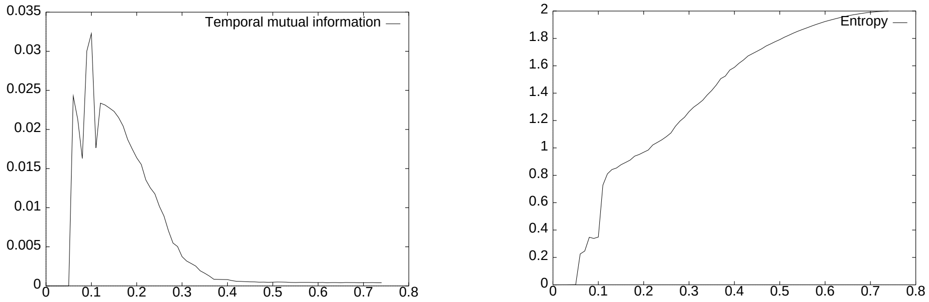


Fig. 4. Temporal mutual information between two sites separated by one time step, and site entropy, both for 4-state 2-neighbor Cellular Automata

The exponential dependence of transient lengths on the size of the CA is analogous to the exponential dependence on problem size in the NP and PSPACE complexity classes. As for the halting-computations, it will be formally undecidable for an arbitrary CA in the vicinity of a phase transition whether transients will ever die out. The increase in mutual information indicates that the correlation length is growing, which implies further evidence for a phase transition in that region. Of course we cannot observe a *real* phase transition other than in the thermodynamic limit. Other discussions for the “Edge of chaos” hypothesis can be found in the work of Crutchfield on continuous dynamical systems [12] and the resulting ϵ-machine reconstruction. In [14] the so-called *intrinsic computation* abilities of a continuous dynamical system are investigated. This term refers to structures that emerge in a system’s behavior that can be understood in computational terms. The output of the system (e.g. an iterative map, $x_{n+1} = f(x_n)$) in time is coarse grained into a sequence of zeros and ones. In other words the output domain x_n is divided into two regions, $P_0 = \{x_n < x_c\}$ and $P_1 = \{x_n \geq x_c\}$, where x_c is an arbitrary chosen division point. The complexity of the dynamical system is quantified by construction of the minimal regular language which accepts the generated sequence. The complexity and entropy (see Eq. 2) for the logistic map was examined in [14] using the method of regular language complexity (size of the corresponding finite automaton). It was found that the lowest values of complexity corresponds to the periodic and fully chaotic regimes of the map. The highest value of the complexity occurs where the period doubling cascade of the map meets the band-merging cascade, i.e. at the border between order and chaos. In [17] the work of Langton and Crutchfield is complemented by examining the dynamical behavior of a well-known computational device: the Turing Machines (TM). A class of 7-state 4-symbol Turing machines, which also includes Minsky’s universal Turing machine [46], was used to address the question whether universal computation is found between order and chaos. A large number of randomly created TM’s was used to generate three different sequences: a sequence of symbols read, a sequence of states and a sequence of moves made by the TM head. For all these sequences, the corresponding regular language complexity was calculated using the technique of ϵ-machine reconstruction and plotted against its block-entropy (see Eq. 4). They found that the most complex TM’s are indeed located at intermediate values of the entropy, including Minsky’s universal TM. Mitchell *et al.*, reviewed this idea of computation at the “edge of chaos” and reported on experiments producing very different results from the original experiment by Packard [54], they suggest that the interpretation of the original results is not correct [48]. Those negative results did not disprove the hypothesis that computational capability can be correlated with phase transitions in CA rule space; they showed only that Packards results did not prove the hypothesis [47]. All in all this is still an open research question that might have a large impact on the understanding of computation in CA’s.

2.4 Modeling with Cellular Automata

Cellular Automata can be an alternative to Differential Equations (DE) for the modeling of physical systems. To integrate a DE numerically, it must be discretized in some way. This discretization is an approximation essentially equivalent to setting up a local discrete (dynamical) system that in the macroscopic limit reduces to the DE under consideration. The idea now is to *start* with a discrete system rather than a continuous description. There is no telling which model (a-priori discrete, or continuous) models the physics best. It is instructive to consider the example of the wave equation in one dimension [1]

$$\frac{\partial^2 f}{\partial t^2} = c^2 \frac{\partial^2 f}{\partial x^2} \,. \tag{9}$$

This equation has two types of solution that are waves traveling to the right and to the left with wave vectors k and frequencies $\omega_k = ck$:

$$f = \sum_k \left(A_k e^{i(kx - \omega_k t)} + B_k e^{i(kx + \omega_k t)} \right) \tag{10}$$

A particular solution is obtained by choosing coefficients A_k and B_k:

$$f = \overline{A}\,(x - ct) + \overline{B}\,(x + ct) \tag{11}$$

with

$$\overline{A}(x) = \sum_k A_k e^{ikx} \tag{12}$$

and $\overline{B}(x)$ analogous, with $\overline{A}(x)$ and $\overline{B}(x)$ two arbitrary functions that specify the initial conditions of the wave in an infinite space. We can construct a simple one-dimensional CA analog to this wave equation. For each update, adjacent cells are paired into partitions of two cells each, where the pairing switches from update to update: the dynamics is completely given by switching the contents of two adjacent cells in an update.

1	−1	−1	1	1	−1	1	1	−1	1	−1	1	1	−1	1	1
−1	1	1	−1	−1	1	1	1	1	−1	1	−1	−1	1	1	1
1	1	1	−1	−1	1	1	1	1	1	−1	−1	−1	1	1	−1
1	1	−1	1	1	−1	1	1	1	1	−1	−1	1	−1	−1	1
1	−1	1	1	1	1	−1	1	1	−1	1	1	−1	−1	−1	1

Given an arbitrarily chosen initial condition, it can be seen that the contents of the odd cells move systematically to the right, where the contents of the even cells move to the left; both with a constant velocity c. The dynamics of this CA is the same as the dynamics of the wave equation in infinite space, we only need to code the initial conditions in the cells appropriately. If a binary representation of the cells is used $\{1, -1\}$, then the local average over the odd cells represents the right traveling wave $\overline{A}(x - ct)$, and the local average over the even cells represents $\overline{B}(x + ct)$.

Another useful CA simulation is found in the study of excitable media. Winfree *et al.* [70] discussed already a simple 2D example that requires three symbols denoting relevant states in a biological cell: Q for quiescent, E for excited, and T for tired. If one of the neighbors is excited, Q is followed by E. After one time step, an excited cell becomes tired and the is set back to Q. This rule gives rise to propagating spirals that are qualitatively similar to those observed in Belousov-Zhabotinsky reactions.

A beautiful extension to this 2D biological cellular model was first reviewed by Celada *et al.* [8]. In this system model, B and T lymphocytes are modeled together with antigen and antibody molecules. They showed simulations of cell response to antigens, response regulation, the minimum number of MHC-type of molecules, and natural selection in the MHC-species repertoire.

One of the most successful efforts to use CAs for simulation of DEs is the lattice gas automaton to simulate hydrodynamics. This will be described in the Section 4. First different parallel execution models for cellular automata will be presented in the next section.

3 Execution Models for Cellular Automata

3.1 Synchronous Cellular Automata versus Asynchronous Cellular Automata

The cellular automata (CA) model is a conceptual simple and effective solver for dynamic complex systems (DCS). From a modelers perspective, a CA model allows the formulation of a DCS application in simple rules. From a computer simulation perspective, a CA model provides an execution mechanism that evaluates the temporal dynamic behavior of a DCS given these simple rules. An important characteristic of the CA execution mechanism is the particular update scheme that applies the rules iteratively to the individual cells of the CA. The different update schemes impose a distinct temporal behavior on the model. Thus we must select the proper update mechanism that aligns with the dynamics of the model.

In the previous discussion, the update mechanism of CAs is described as being synchronously in parallel. However, for certain classes of DCS, the temporal dynamic behavior is asynchronously. In particular, systems with heterogeneous spatial and temporal behavior are, in general, most exactly mapped to asynchronous models [3,41]. In case asynchronous models are solved by CA, the asynchronous temporal behavior must be captured by the update mechanism. This class of CA is called Asynchronous Cellular Automata (ACA) [23,39,51,53]. The ACA model incorporates asynchronous cell updates, which are independent of the other cells, and allows for a more general approach to CA. With these qualifications, the ACA is able to solve more complicated problems, closer to reality [75].

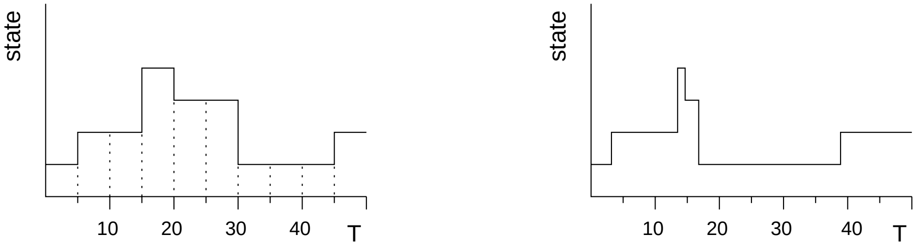
Dynamic systems with asynchronous updates can be forced to behave in a highly inhomogeneous fashion. For instance in a random iteration model it is assumed that each cell has a certain probability of obtaining a new state and that cells iterate independently. As an example one can think of the continuous-time probabilistic dynamic model for an Ising spin system [40].

3.2 Types of Simulation

Essential to every model is the time base on which changes to the system state occur. Accordingly, models can be classified depending on their temporal dynamic behavior [74]. A model is a *continuous-time* model when time flows smoothly and continuously. A model is a *discrete-time* model if time flows in jumps of some specified time unit.

Continuous time models can be further divided into *differential equation* and *discrete-event* classes. A differential equation model is a continuous-time model where changes in the state occur smoothly and continuously in time. In a discrete-event model, even though time flows continuously, state changes can occur only at discrete points in time: time jumps from one event to the next, while these events can occur arbitrarily separated from each other.

By the very nature of the CA rules that define the state transformations, the temporal dynamic behavior classes that are applicable to cellular automata are discrete-time and discrete-event. The figures Fig. 5(a) and Fig. 5(b) show the differences between the temporal behavior of state changes in both classes.



(a) State changes in a discrete-time system model

(b) State changes in a discrete-event system model

Fig. 5. Temporal behavior of discrete-time and discrete-event system model

3.2.1 Discrete-Time Models and Time-Driven Simulation In discrete-time models the progress of time is modeled by time advances of a fixed increment, for example time is advanced in increments of exactly Δt time units. The execution mechanism that implements this temporal dynamic behavior is called *time-driven* simulation, since the clock initiates the state transitions of each individual cell in the CA. The execution mechanism in time-driven simulation is characterized by an iterative loop that, after each update of the simulation time, updates the state variables for the next time interval $\langle t, t + \Delta t]$. The new state of a cell at time $t + \Delta t$ is calculated from the state of the cell and its neighbors at time t.

Time-driven simulation is the most widely known and applied approach for the simulation of CA models and natural systems in general. However, with the usage of time-driven simulation one has to ascertain that the time step Δt is small enough to capture every state change in the system model. This might imply that we need to make Δt arbitrarily small, which is certainly not acceptable with respect to the computational requirements involved. Therefore, time-driven simulation is less appropriate for the simulation of discrete-event models, as there may be many clock ticks in which no events occur.

3.2.2 Discrete-Event Models and Event-Driven Simulation The progress of time in discrete-event models is modeled by the occurrence of instantaneous state changes, called events, at random times and independent from each other. The execution mechanism that implements this temporal dynamic behavior is called *event-driven* simulation. In event-driven simulation, the progress of simulation time depends on the occurrence of the next event. The event-driven simulation execution mechanism maintains an ordered event list to hold expected future events. The simulation time progresses from the current time to the next scheduled event time. The simulation of one event may generate new events that are scheduled for future execution.

An elegant and efficient characteristic of the event-driven simulation approach is that periods of inactivity are skipped over by advancing the simulation clock from event time to the next event time. This is perfectly save since—by definition—all state changes only occur at event times. Therefore causality and the validity of the simulation is guaranteed. The event-driven approach to discrete systems is usually exploited in queuing and optimization problems. However, as we will see next, it is also a paradigm for the simulation of continuous-time systems.

3.3 Parallel Simulation of Cellular Automata Models

The parallelization of the CA, both for synchronous and asynchronous models, is realized by geometric decomposition. That is, the individual cells of the CA are aggregated into sublattices, which are mapped to the parallel processors. However, the parallel synchronization mechanism between the sublattices are very different for synchronous and asynchronous CA models.

3.3.1 Parallel Synchronous Cellular Automata Simulation Similar to the sequential execution of synchronous CA, the cells in a parallel synchronous CA simulation undergo simultaneous state transitions under direction of a global clock. All cells must finish their state transition computations before any cell can start simulating the next clock tick.

The parallelization of the discrete-time simulation is achieved by imitating the synchronous behavior of the simulation. The simulation is arranged into a sequence of rounds, with one round corresponding to one clock tick. Between each round a global synchronization of all cells indicates that the cells have finished their state change at time step t and the new time step $t + \Delta t$ can be started.

Generally, the simulation proceeds in two phases, a computation and state update phase, and a communication phase. The progression of time in time-driven simulation is illustrated in Fig. 6.

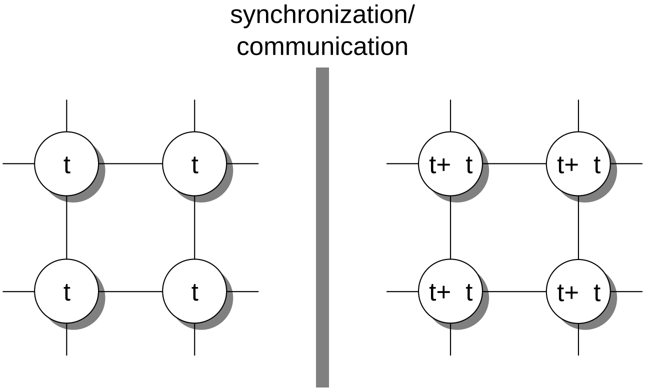


Fig. 6. Time-driven simulation of a synchronous CA model, where computation and communication phases succeed each other

3.3.2 Parallel Asynchronous Cellular Automata Simulation In parallel ACA simulation, state transitions (further called events) are not synchronized by a global clock, but rather occur at irregular time intervals. In these simulations few events occur at any single point in simulated time and therefore parallelization techniques based on synchronous execution using a global simulation clock perform poorly. Concurrent execution of events at different points in simulated time is required, but this introduces severe synchronization problems. The progress of time in event-driven simulation is illustrated in Fig. 7.

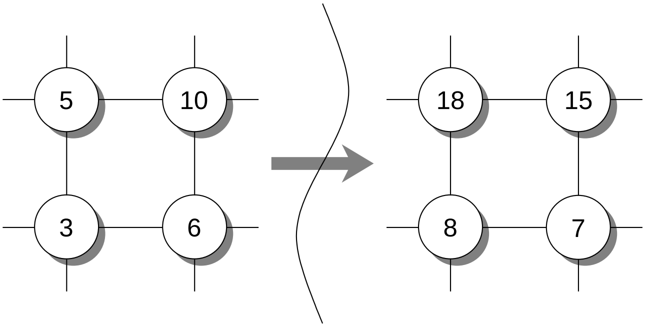


Fig. 7. Progress of simulation time in event-driven simulation. As the cells evolve asynchronously in time, the simulation time of the individual cells are different

The absence of a global clock in asynchronous execution mechanisms necessitates sophisticated synchronization algorithms to ensure that cause-and-effect relationships are correctly reproduced by the simulator. Parallel discrete event simulation is essentially concerned with the correct ordering, or scheduling, of the asynchronous execution of events over parallel processors. There are basically two methods to impose the correct temporal order of the asynchronous event execution: conservative and optimistic methods.

First, the *conservative* approach proposed by Chandy and Misra [9] strictly imposes the correct temporal order of events. Second, the *optimistic* approach, introduced by Jefferson [24], uses a detection and recovery mechanism: whenever the incorrect temporal order of events is detected, a rollback mechanism is invoked to recover. Although both approaches have their specific application area, optimistic methods offer the greatest potential as a general-purpose simulation mechanism.

In optimistic simulation, the parallel simulation processes execute events and proceed in local simulated time as long as they have any input at all. A consequence of the optimistic execution of events is that the local clock or Local Virtual Time (LVT) of a process may get ahead of its neighbors' LVTs, and it may receive an event message from a neighbor with a simulation time smaller than its LVT, that is, in the past of the simulation process. The event causing the causality error is called a *straggler*. If we allow causality errors to happen, we must provide a mechanism to recover from these errors in order to guarantee a causally correct parallel simulation. Recovery is accomplished by undoing the effects of all events that have been processed prematurely by the process receiving the straggler. The net effect of the recovery procedure is that the simulation process rolls back in simulated time.

The premature execution of an event results in two things that have to be rolled back: (i) the state of the simulation process and (ii) the event messages sent to other processes. The rollback of the state is accomplished by periodically saving the process state and restoring an old state vector on rollback: the simulation process sets its current state to the last state vector saved with simulated time earlier than the timestamp of the straggler. Recovering from premature sent messages is accomplished by sending an *anti-message* that annihilates the original when it reaches its destination. The messages that are sent while the process is propagating forward in simulated time, and hence correspond with simulation events, are called *positive messages*.

A direct consequence of the rollback mechanism is that more anti-messages may be sent to other processes recursively, and that it allows all effects of erroneous computation to be eventually canceled. As the smallest unprocessed event in the simulation is always safe to process, it can be shown that this mechanism always makes progress under some mild constraints [24].

In optimistic simulation the notion of global progress in simulated time is administered by the Global Virtual Time (GVT). The GVT is the minimum of the LVTs for all the processes and the timestamps of all messages (including anti-messages) sent but unprocessed. No event with timestamp smaller than the GVT will ever be rolled back, so storage used by such event (i.e., saved state vector and event message) can be discarded. Also, irrevocable operations such as I/O cannot be committed before the GVT sweeps past the simulation time at which the operation occurred. The process of reclaiming memory and committing irrevocable operations is referred to as *fossil collection*.

To summarize, the parallel *synchronous* execution mechanism for discrete-time models mimics the sequential synchronous execution mechanism by interleaving a computation and state update phase with a synchronization and communication phase. The parallel execution mechanism is fairly simple and induces a minimum of overhead on the computation. The parallel *asynchronous* execution mechanism for discrete-event models, the so-called optimistic simulation method, is more expensive than its sequential counterpart. The synchronization mechanism in optimistic simulation requires extra administration, such as state saving and rollback. Despite this overhead, optimistic simulation is an efficient parallel execution mechanism for discrete-event models. In Section 5, two applications are presented that are typical examples of respectively synchronous and asynchronous CA models.

4 Cellular Automata as Models for Fluid Flow

4.1 Introduction

Section 2 introduced the basic idea behind Cellular Automata (CA) and exemplified how one can reason about information and complexity in general CAs. Here we introduce a very specific CA, which, as will become clear later on, can be used as a model of fluid flow. This class of CA is called Lattice Gas Automata (LGA), and they are described in detail in two recent books [11,58].

Suppose that the state of a cell is determined by b_m surrounding cells. Usually, only the nearest and next-nearest neighbors are considered. For example, on a square lattice with only nearest neighbor interactions $b_m = 4$, if next-nearest neighbors are also included $b_m = 8$, and on a hexagonal lattice with nearest neighbor interactions $b_m = 6$. Furthermore, suppose that the state of the cell is a vector $\mathbf{n}$ of $b = b_m$ bits. Each element of the state vector is associated with a direction on the CA lattice. For example, in the case of a square grid with only nearest neighbor interactions we may associate the first element of the state vector with the *north* direction, the second with *east*, the third with *south* and the fourth with *west*. With these definitions we construct the following CA rule (called the LGA rule), which consists of two sequential steps:

1. Each bit in the state vector is moved in its associated direction (so in the example, the bit in element 1 is moved to the neighboring cell in the north) and placed in the state vector of the associated neighboring cell, in the same position (so, the bit in element 1 is moved to element 1 of the state vector in the cell in the north direction). In this step each cell is in fact moving bits from its state vector in all directions, and at the same time is receiving bits from all directions, which are stored into the state vector.

2. Following some deterministic or stochastic procedure, the bits in the state vector are reshuffled. For instance, the state vector $(1, 0, 1, 0)$ is changed to $(0, 1, 0, 1)$.

As a refinement, one may also introduce b_r extra bits in the state vector which, as it were, reside on the cell itself, i.e. are not moved to another cell in step 1 of the LGA rule. In that case the length of the state vector $b = b_m + b_r$. These b_r residing bits do however participate in the reshuffling step 2.

It is clear that the class of LGA-CA that we have just defined is very large. We have freedom to choose the CA lattice, the interaction list, the number of residing bits, and the reshuffling rule. Once all this is done, we may expect that the specific LGA-CA that we defined has a very rich dynamical behavior, depending on the initial conditions and the size of the grid. Except maybe for 1-dimensional lattices, a detailed study of the dynamics of such CA is probably not feasible. It was shown by Moore and Nordhal [49] that the problem of LGA prediction is P-complete, thus cannot be solved in parallel in polylogarithmic time. This implies that the only way out is a step by step explicit simulation. Our new CA therefore seems like a nice toy that may exhibit a very complex dynamical behavior, but no more than that. However, maybe very surprisingly, if we associate physical quantities to our CA, enforce physical conservation laws on the bit reshuffling rule of step 2, and use methods from theoretical physics to study the dynamics, we *are* in fact able to analyze the CA in terms of its *average* behavior, i.e. the average state vector of a cell and the average flow of bits between cells can be calculated. Even better, it turns out, again within the correct physical picture, that this CA behaves like a real fluid (such as water) and therefore can be used as a model for hydrodynamics. Furthermore, as the LGA rule is intrinsically local (only nearest and next-nearest neighbor interactions) we constructed an inherently parallel model for fluid flow.

4.2 Associating Physics with the LGA-CA

Our current image of the LGA-CA is that of bits that first move from a cell to a neighboring cell and are then reshuffled into another direction. Now we associate the bits in the state vector with *particles*; a one-bit codes for the presence of a particle, and a zero bit codes for the absence of a particle. Assume that all particles are equal and have a mass of 1. Step 1 in the LGA-CA is now interpreted as streaming of particles from one cell to another. If we also introduce a length scale, i.e. a *distance* between the cells (usually the distance between nearest neighbors cells is taken as 1) and a time scale, i.e. a *time duration* of the streaming (i.e. step 1 in the LGA-CA rule, usually a time step of 1 is assumed), then we are able to define a *velocity* $\mathbf{c}_i$ for each particle in direction i (i.e. the direction associated with the i-th element of the state vector $\mathbf{n}$). Step 1 of the LGA-CA is the streaming of particles with velocity $\mathbf{c}_i$ from one cell to a neighboring cell. The residing bits can be viewed as particles with a zero velocity, or rest particles. Now we may imagine, as the particles meet in a cell, that they collide. In this collision the velocity of the particles (i.e. both absolute speed and direction) can be changed. The reshuffling of bits in step 2 of the LGA-CA rule can be interpreted as a collision of particles. In a real physical collision, mass, momentum, and energy are conserved. Therefore, if we formulate the reshuffling such that these three conservation laws are obeyed, we have constructed a true Lattice Gas Automaton, i.e. a gas of particles that can have a small set of discrete velocities $\mathbf{c}_i$, moving in lock-step over the links of a lattice (space is discretized) and that all collide with other particles arriving at a lattice point at the same time. In the collisions, particles may be send in other directions, in such a way that the total mass and momentum in a lattice point is conserved.

We can now associate with each cell of the LGA-CA a density ρ and momentum $\rho\mathbf{u}$, with $\mathbf{u}$ the velocity of the gas:

$$\rho = \sum_{i=1}^{b} N_i, \tag{13}$$

$$\rho\mathbf{u} = \sum_{i=1}^{b} \mathbf{c}_i N_i, \tag{14}$$

where $N_i = \langle \mathbf{n}_i \rangle$, i.e. a statistical average of the Boolean variables; N_i should be interpreted as a particle density.

The good thing now is, if we let the LGA evaluate and calculate the density and momentum as defined in Eqs. (13, 14), that these quantities behave just like a real fluid.

In Figs. 8A and 8B we show an example of an LGA simulation of flow around a cylinder. In Fig. 8A we show the results of a single iteration of the LGA, so in fact we have assumed that $N_i = \mathbf{n}_i$. Clearly, the resulting flow field is very noisy. In order to arrive at smooth flow lines one should calculate $N_i = \langle \mathbf{n}_i \rangle$. Because the flow is static, we calculate N_i by averaging the boolean variables $\mathbf{n}_i$ over a large number of LGA iterations. The resulting flow velocities are shown in Fig. 8B.

The evolution of the boolean variables $\mathbf{n}_i$ can be expressed as

$$\mathbf{n}_i\left(\mathbf{x}+\mathbf{c}_i, t+1\right) - \mathbf{n}_i\left(\mathbf{x}, t\right) = \mathit{\Delta}_i\left(\mathbf{n}\left(\mathbf{x}, t\right)\right) \tag{15}$$

where $\mathbf{x}$ denotes the position of a lattice point and $\mathit{\Delta}$ is the collision operator. The collision operator must obey mass, momentum, and energy conservation, i.e.

$$\sum_{i=1}^{b} \mathit{\Delta}_i\left(\mathbf{n}\right) = 0, \tag{16}$$

$$\sum_{i=1}^{b} \mathbf{c}_i \mathit{\Delta}_i\left(\mathbf{n}\right) = 0, \tag{17}$$

$$\sum_{i=1}^{b} c_i^2 \mathit{\Delta}_i\left(\mathbf{n}\right) = 0, \tag{18}$$

where $c_i = |\mathbf{c}_i|$. One can ask if the evolution equation 15 is also valid for the averaged particle densities N_i. It turns out that this is possible, but only under

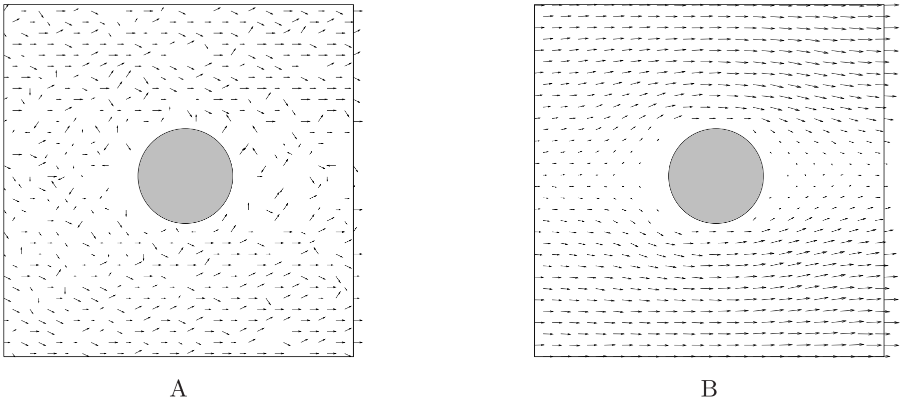


Fig. 8. A. LGA simulation of flow around a cylinder, the result of a single iteration of the LGA is shown. The arrows are the flow velocities, the length is proportional to the absolute velocity. The simulations were done with FHP-III, on a 32 × 64 lattice, the cylinder has a diameter of 8 lattice spacings, only a 32 × 32 portion of the lattice is shown; periodic boundary conditions in all directions are assumed.
B. As in Fig. 8A, now the velocities are shown after averaging over 1000 LGA iterations

the Boltzmann molecular chaos assumption which states that particles that collide are not correlated before the collision, or, in equations, that for any number of particles k, $\langle n_1 n_2 \ldots n_k \rangle = \langle n_1 \rangle \langle n_2 \rangle \ldots \langle n_k \rangle$. In that case one can show that $\langle \Delta_i (\mathbf{n}) \rangle = \Delta_i (\mathbf{N})$. By averaging Eq. (15) and applying the molecular chaos assumption we find

$$N_i (\mathbf{x} + \mathbf{c}_i, t + 1) - N_i (\mathbf{x}, t) = \Delta_i (\mathbf{N} (\mathbf{x}, t)) \ . \tag{19}$$

A first order Taylor expansion of $N_i (\mathbf{x} + \mathbf{c}_i, t + 1)$, substituted into Eq. (19) results in

$$\partial_t N_i (\mathbf{x}, t) + \partial_\alpha \mathbf{c}_{i\alpha} N_i (\mathbf{x}, t) = \Delta_i (\mathbf{N} (\mathbf{x}, t)) \tag{20}$$

Note that the shorthand ∂_t means $\frac{\partial}{\partial t}$, the subscript α denotes the α-component of a D-dimensional vector, where D is the dimension of the LGA lattice, and that we assume the Einstein summation convention over repeated Greek indices (e.g. in two dimensions $\partial_\alpha \mathbf{c}_{i\alpha} N_i = \partial_x \mathbf{c}_{ix} N_i + \partial_y \mathbf{c}_{iy} N_i$). Next we sum Eq. (20) over the index i and apply Eqs. (13, 16, 17), thus arriving at $\partial_t \rho + \partial_\alpha (\rho \mathbf{u}_\alpha) = 0$, or

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \mathbf{u} = 0, \tag{21}$$

which is just the equation of continuity that expresses conservation of mass in a fluid. One can also first multiply Eq. (20) with $\mathbf{c}_i$ and then summate over the index i. In that case we arrive at

$$\partial_t \rho u_\alpha + \partial_\beta \Pi_{\alpha\beta} = 0, \tag{22}$$

with

$$\Pi_{\alpha\beta} = \sum_{i=1}^{b} \mathbf{c}_{i\alpha} \mathbf{c}_{i\beta} \mathbf{N}_i. \tag{23}$$

The quantity $\Pi_{\alpha\beta}$ must be interpreted as the flow of the α-component of the momentum into the β-direction, $\Pi_{\alpha\beta}$ is the momentum density flux tensor. In order to proceed one must be able to find expressions for the particle densities N_i. This is a highly technical matter that is described in detail in e.g. [11,58]. The bottom line is that one first calculates the particle densities for a LGA in equilibrium, N_i^0, and substitute them into Eq. (23). This results in an equation that is almost similar to the Euler equation, i.e. the expression of conservation of momentum for an inviscid fluid. Next, one proceeds by taking into account small deviations from equilibrium, resulting in viscous effects. Again, after a very technical and lengthy derivation one is able to derive the particle densities, substitute everything into Eq. (23) and derive the full expression for the momentum conservation of the LGA, which again very closely resembles the Navier-Stokes equations for an incompressible fluid. The viscosity and sound speed of the LGA are determined by its exact nature (i.e. the lattice, the interaction list and the number of residing particles, and the exact definition of the collision operator).

At first sight the average, macroscopic behavior of the LGA may come as a big surprise. The LGA-CA is a model that reduces a real fluid to one that consists of particles with a very limited set of possible velocities, that live on the links of a lattice and all stream and collide at the same time. Yet, theoretical analysis and a large body of simulation results show that, although the LGA is indeed a very simple model, it certainly is a realistic model of a real fluid. However, it is true that not all LGA behave as a real fluid. The underlying lattice must have enough symmetry such that the resulting macroscopic equations are isotropic, as in a real fluid. For instance, the first LGA, the so-called HPP model, which is defined on a two dimensional square lattice with only nearest-neighbor interactions and no rest particles, is not isotropic. The FHP models, which have a two dimensional hexagonal lattice, do possess enough symmetry and their momentum conservation law have the desired isotropy property.

To end this section we stress once more that the LGA is an intrinsically local CA and therefore gives us an inherently parallel model for fluid flow simulations. Some case studies to support this will be provided in later sections.

4.3 The FHP Model

The FHP model, named after its discoverers Frisch, Hasslacher, and Pomeau, was the first LGA with the correct (isotropic) hydrodynamic behavior. The FHP model is based on a 2-dimensional hexagonal lattice, as in Fig. 9. This figure

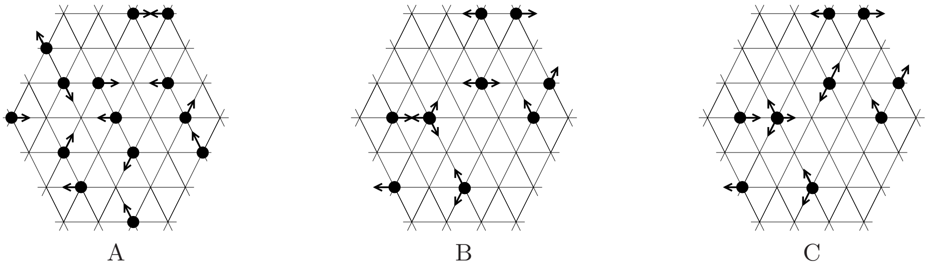


Fig. 9. Lattice and update mechanism of the FHP-I LGA. A dot denotes a particle and the arrow its moving direction. In A to C the propagation and collision phases are shown for some initial configuration

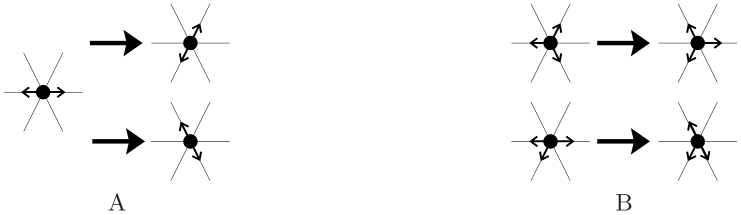


Fig. 10. Collision rules of FHP-I. A dot denotes a particle and the arrow its moving direction. The left figure shows the two particle collisions, the right figure the three particle collisions

also shows examples of streaming and collisions of particles in this model. In the FHP-I model, which has no rest particles (i.e. $b_r = 0$ and $b = b_m = 6$), only 2-body and 3-body collisions are possible, see Fig. 10. Note that all these collision configurations can off course be rotated over multiples of 60°.

For this model we can easily write an explicit expression for the collision operator, as

$$\varDelta_i(\mathbf{n}) = \varDelta_i^{(3)}(\mathbf{n}) + \varDelta_i^{(2)}(\mathbf{n}) \; . \tag{24}$$

The three body collision operator is

$$\varDelta_i^{(3)}(\mathbf{n}) = n_{i+1}n_{i+3}n_{i+5}\overline{n}_i\overline{n}_{i+2}\overline{n}_{i+4} - n_i n_{i+2}n_{i+4}\overline{n}_{i+1}\overline{n}_{i+3}\overline{n}_{i+5}, \tag{25}$$

where $\overline{n}_i = 1 - n_i$ and the subscript should be understood as “modulo 6”. A similar expression can be obtained for the two-body collisions (see e.g. [58]). It is clear that the implementation of this LGA, i.e. the evolution equation 15 with the FHP-I collision operator (Eq. 24), using bit wise operations, is straightforward and can result in very fast simulations with low memory consumption. Furthermore, the inherent locality of the LGA rule makes parallelization trivial. Next, by averaging the boolean variables $\mathbf{n}_i$, either in space or in time, one obtains the particles densities N_i and from that, using Eqs. (13, 14), the density and fluid velocity. Many people, especially those who are used to simulate flow patterns based on numerical schemes derived from the Navier Stokes equations, find it hard to believe that such a simple Boolean scheme is able to produce realistic flow simulations. Yet the LGA, which in a sense originated from the original ideas of von Neuman who invented CA as a possible model to simulate life, is a very powerful and viable model for hydrodynamics.

4.4 The Lattice Boltzmann Method

Immediately after the discovery of LGA as a model for hydrodynamics, it was criticized on three points; noisy dynamics, lack of Galilean invariance, and exponential complexity of the collision operator. The noisy dynamics is clearly illustrated in Fig. 8A. The lack of Galilean invariance is a somewhat technical matter which results in small differences between the equation for conservation of momentum for LGA and real Navier Stokes equations, for details see [58]. Adding more velocities in an LGA leads to increasingly more complex collision operators, exponentially in the number of particles. Therefore, another model, the Lattice Boltzmann Method (LBM), was introduced. This method is reviewed in detail in [10].

The basic idea is that one should not model the individual particles $\mathbf{n}_i$, but immediately the particle densities N_i, i.e. one iterates the Lattice Boltzmann Equation 19. This means that particle densities are streamed from cell to cell, and particle densities collide. This immediately solves the problem of noisy dynamics. However, in a strict sense we no longer have a CA with a boolean state vector. However, we can view LBM as a generalized CA. By a clever choice of the equilibrium distributions N_i^0 the model becomes isotropic and Galilean invariant, thus solving the second problem of LGA. Finally, a very simple collision operator is introduced. This so-called BGK collision operator models the collisions as a single-time relaxation towards equilibrium, i.e.

$$\Delta_i^{BGK}(\mathbf{N}) = \tau(N_i^0 - N_i). \tag{26}$$

Eqs. (19, 26) together with a definition of the equilibrium distributions result in the Lattice-BGK (L-BGK) model. The L-BGK model leads to correct hydrodynamic behavior. The viscosity, of a two-dimensional L-BGK on a hexagonal lattice is given by:

$$\nu = \frac{1}{4}\left(\frac{1}{\tau} - \frac{1}{2}\right) . \tag{27}$$

The L-BGK is also developed for other lattices, e.g. in two or three dimensions cubic lattices with nearest and next nearest neighbor interactions. The LBM, and especially the L-BGK has found widespread use in simulations of fluid flow.

4.5 Parallelism and Applications

The LGA and LBM have been used to simulate many types of flow in, especially, complex geometries. In Section 5.1 we show in detail such an application. Here we will further discuss parallelism in LGA and LBM, and show some examples of applications of large scale parallel LGA and LBM simulations.

The local nature of the LGA and LBM interactions allows a very straightforward realization of parallelism. A simple geometric decomposition of the lattice with only local message passing between the boundaries of the different domains is sufficient to realize an efficient parallel simulation. For instance, we have developed a generic 2-dimensional LGA implementation that is suitable for any multi-species (thermal) LGA [16]. Parallelism was introduced by means of a 1-dimensional, i.e. strip-wise, decomposition of the lattice. As long as the grid dimension compared to the number of processors is large enough, this approach results in a very efficient parallel execution. This LGA system is mainly used for simulations in simple rectangular domains without internal obstacles. However, in a more general case, where the boundaries of the grid have other forms and internal structure (i.e. solid parts where no particles will flow) the simple strip-wise decomposition results in severe load imbalance. In this case, as was shown by Kandhai *et al.* [30], a more advanced decomposition scheme, the Orthogonal Recursive Bisection (ORB) method, still leads to highly efficient parallel LBM simulations. ORB restores the load balancing again, however at the price of a somewhat more complicated communication pattern between processors.

As in many models, the specification of initial and boundary methods turns out the be much more difficult than anticipated. The same is true for LGA and LBM. Solid walls are usually implemented using a bounce back rule (i.e. sending a particle back into the direction where it came from) thus implementing a no-slip boundary. Kandhai *et al.* have investigated this bounce back rule in detail for L-BGK [31] and conclude that this simple rule, although it may have a negative effect on the accuracy of the L-BGK, is a very suitable boundary condition in simulations, as long as one is not interested in the details of the flow close to the boundaries. Khandai *et al.* also investigated several formulations for initial conditions and other types of boundary conditions (e.g. specifying a certain velocity on a boundary).

As an example of a large scale parallel L-BGK simulation we refer to Ref. [34], where flow in a random fibrous network, as a model for paper, was simulated. The permeability that was obtained from the simulations was in very good agreement with experimental results. Another impressive example is flow in a Static Mixer Reactor [32]. Here, L-BGK simulations and conventional Finite Element simulations where compared, which agreed very well. The simulation results also agree very well with experimental results. This shows that L-BGK, which is much easier to parallelize and much easier to extend with more complex modeling compared to FE, (multi-species flow, thermal effects, reactions), is very suitable in real life problems involving complex flow.

5 Selected Applications

5.1 Modeling Growth and Form in a Moving Fluid Using *Synchronous* Cellular Automata

The basic idea of modeling growth and form of marine sessile suspension feeders, as for example sponges and stony corals [28,29], will be briefly discussed in the next section. The simulated growth forms will be only qualitatively discussed, more detailed quantitative measurements on for example the space-filling properties, expressed in fractal dimensions, centers of gravity of the simulated objects, and absorption measurements are presented elsewhere [28,29]. In the model both the parallelism present in physical environment (dispersion of nutrients through diffusion and flow) and the parallelism present in the growth process, will be exploited. The dispersion of nutrients will be modeled using the lattice Boltzmann method discussed in Section 4.4, while the growth of the stony coral will be modeled using a probabilistic cellular automaton.

5.1.1 Biological Background Many marine sessile suspension feeders from various taxonomic groups, as for example sponges, hydrocorals, and stony corals, exhibit a strong morphological plasticity, which is in many cases related to the impact of hydrodynamics. The exposure to water movement represents one of the dominant environmental parameters. In a number of cases it is possible to arrange growth forms of sponges, hydrocorals, and stony corals along a gradient of the amount of water movement [27]. In the examples of stony corals, the growth form gradually transforms from a compact shape under exposed conditions, to a thin-branching one under sheltered conditions. In Fig. 11 two extreme examples of growth forms are shown of *Pocillopora damicornis*. Form A originates from a site sheltered to water movement, while B was collected from an exposed site. Between both extremes, a range of gradually changing intermediate growth forms exist [67].

Stony corals are typical modular organisms. Modular growth is defined as the growth of genetic identical individuals by repeated iteration of (multi-cellular) parts: the modules [22]. Modules might be the polyp of a coral or for instance a shoot with an apical meristem in seed plants. The modular growth of stony corals is relatively simple when compared to more complex modular organisms like seed plants. The modular growth of these organisms can be defined as parallel modular growth, where the various modules grow almost independently, only limited by steric hindrance. Because of the almost independently growing modules, which are not limited by the development of other modules, some important simplifications can be made in the modeling of the growth process. The organisms can increase in size without a decrease in growth velocities, where growth of these organisms will be limited by external factors like strong water movements.

To obtain insight into the influence of hydrodynamics on the growth process of sessile suspension feeders, a morphological simulation model was developed. In the absence of flow, the distribution of nutrients around the growth form can be modeled as a diffusion process in a steady state: there is a source of

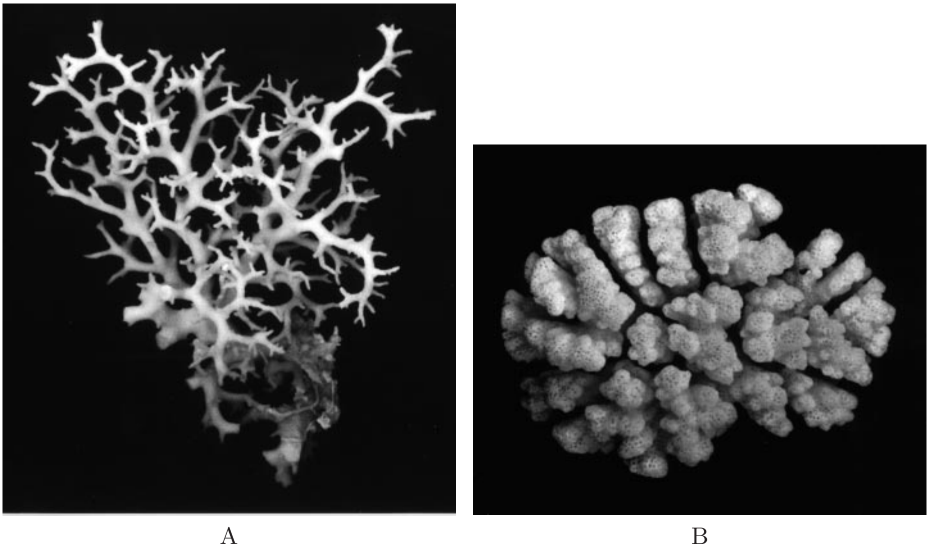


Fig. 11. Growth forms of the stony coral *Pocillopora damicornis*. Form A originates from a site sheltered to water water movement, form B originates from an exposed site

suspended material and the organism continuously consumes nutrients from its environment. In general in a marine environment, there will be a significant contribution of the hydrodynamics to the dispersion pattern of the suspended material around the growth form. In this case the distribution of nutrients around the organism will be determined by a combination of flow and diffusion.

5.1.2 A CA model of Flow and Nutrient Distributions The flow pattern around the simulated growth form was computed by applying the lattice Boltzmann method (see Section 4.4) in combination with a tracer step to study the dispersion of "nutrient" in the system. In these simulations the nutrient particles are dispersed by the combined process of flow and diffusion. With this method simulated growth processes can be studied for various Péclet numbers Pe defined as:

$$Pe = \frac{\overline{u}l}{D} \tag{28}$$

where $\overline{u}$ is the mean flow velocity, l a characteristic length, and D the diffusion coefficient of the nutrient. The contribution of flow to the nutrient distribution can be quantified by the Péclet number. A low value indicates that particles move mainly by diffusion (no influence of hydrodynamics, diffusion dominates) and a high value indicates that their motion is dominated by flow.

Two types of boundary conditions are used: at the borders of the lattice periodic boundary conditions are applied, while in the nodes adjacent to the nodes representing the simulated growth form, solid boundary conditions are used. Periodic boundary conditions can be implemented by exchanging the n_i's of the links at the borders of the lattice. Solid boundary conditions can be represented by exchanging the n_i's between the adjacent node and a neighboring fluid node.

After a lattice Boltzmann iteration, a tracer step is applied where populations of tracer particles are released from source nodes, while the tracer particles are absorbed by the sink nodes: the nodes adjacent to the growth form. The tracer particles can travel from a node at site r in the lattice to one of the 18 adjacent nodes $r + c_i$, where the Péclet number Eq. (28) determines if flow or diffusion dominates. When flow dominates, most particles will move in the direction of the governing flow, while under diffusion dominated conditions the amount of particles which travels in all 18 directions will be about equal. In the simulations the diffusion coefficient D varies, and $\overline{u}$ is kept constant by adjusting the driving force F of the system. Due to the growth of the object the velocity in the free fluid would gradually decrease if the driving force is not adjusted. For details on the computational model we refer to [28].

5.1.3 A CA Model of the Growth Process The growth process is represented by a probabilistic cellular automaton in a similar way as done in the Diffusion Limited Aggregation model [71]. In [42] and [43] it is demonstrated that the Diffusion Limited Aggregation growth model is P-complete for which fast parallel algorithms probably do not exist, the only available option to study this type of growth processes is through explicit simulation. The basic construction of the aggregate is shown in Fig. 12. The cluster is initialized with a “seed” positioned at the bottom. In both the cluster and substrate sites solid boundary conditions are applied. Two flow regimes were studied in the simulations:

1. growth of the aggregates in a mono directional flow;
2. growth of the aggregates in a bidirectional (alternating) flow.

The flow velocity in the system is kept at a low value, all simulations are done in the laminar flow regime. Tracer particles are released from the source plane. The tracer particles are absorbed by the fluid nodes adjacent to obstacle nodes, which can be nodes in the substrate plane or the aggregate nodes. In the growth model it is assumed that both the tracer distribution and flow velocities are in equilibrium and the growth velocity of the aggregate is much slower than the dispersion of the tracer. In the sink nodes the amount of absorbed tracer particles is determined and a new node is added to the aggregate. The probability p that k, an element from the set of open circles $\circ$ (the adjacent sink nodes) will be added to the set of black circles (the aggregate nodes) is given by

$$p(k \in \circ \rightarrow k \in \bullet) = \frac{(a_k)}{\sum_{j \in \circ}(a_j)}\ , \tag{29}$$

where a_k is the absorbed amount of tracer particles at position k.

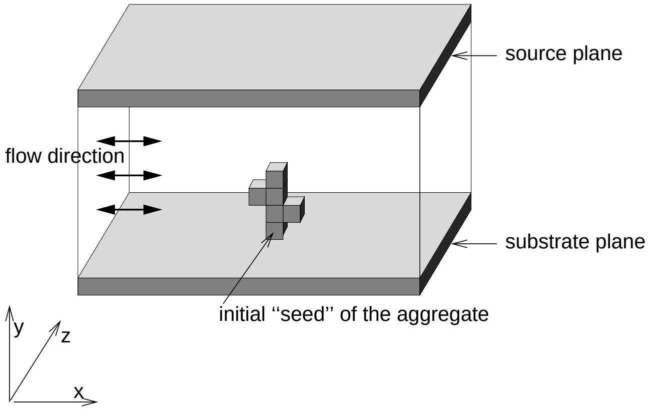


Fig. 12. Basic construction of the aggregate

In the bidirectional flow simulations the flow direction is reversed after each growth step. The aggregation model is summarized in pseudo-code below:

```
initialize aggregate
initialize flow direction

do {
    compute flow velocities until equilibrium;
    compute tracer distribution until equilibrium;
    compute probabilities $p$ that nodes neighboring to the aggregate
        nodes will be added to the aggregate;
    select randomly with probability $p$ one of the growth candidates
        and add it to the aggregate;
    for the bidirectional case: reverse flow direction;
} until ready
```

5.1.4 Parallelization of the CA models We performed the simulations on a lattice consisting of 144^3 sites. The algorithm was implemented in parallel and the simulations were carried out on 16 nodes of a distributed memory Parsytec CC/40 system (approximately 6 Gflops/s). In this parallel implementation the nearest neighbor locality, present in both the lattice Boltzmann step, tracer calculation and growth model are exploited. In the parallel implementation the 144^3 lattice is decomposed into a number of sublattices which are distributed over the processors. The main computation steps (lattice Boltzmann and tracer calculation) are done in the fluid nodes, while only in the growth step some computation is required in the aggregate nodes. Due to the growth of the aggregate a straight forward decomposition (for example partitioning of the lattice in equal sized slices or boxes) would lead to a strong load imbalance. To solve this problem we have tested two strategies to obtain a more equal distribution of the load over the processors:

1. Box decomposition in combination with scattered decomposition.
2. Orthogonal Recursive Bisection (ORB) in combination with scattered decomposition.

In the box decomposition method the lattice is partitioned in 2D in equal sized boxes. In the ORB method [61] the object (the aggregate) is split into two subdomains perpendicular with respect to the xy-plane; after this similar splits are made for respectively the yz-plane and xz-plane. This method is recursively repeated for each subdomain. In the scattered decomposition method [50,57] blocks of data are scattered over the processors. The original lattice is divided by using a partitioning method. These blocks are randomly scattered over the processors, where each processor has the same number of blocks. An example of a scattered decomposition over 4 processors of an irregular shaped object in a 2D lattice is shown in Fig. 13. In this example the lattice is divided into 100 blocks, where each block is randomly assigned to one of the four processors. Most of the computation is done in the blocks containing exclusively fluid nodes. The scattering of the blocks over the processors leads to a spatial averaging of the load, where decreasing block sizes cause a better load balancing and an increasing communication overhead [57]. Especially in simulations in which the shape of the object cannot be predicted, is scattered decomposition an attractive option to improve the load balance in parallel simulations [42,43]. We have compared both decomposition strategies by computing the Load balancing efficiency:

$$\text{Load balancing efficiency} = \frac{l_{min}}{l_{max}} \tag{30}$$

where l_{min} is the load of the slowest process and l_{max} the load of the fastest process. The two decomposition strategies were tested by using two extreme morphologies of the aggregates: a very compact shape and a dendritic shaped aggregate and by measuring the load balancing efficiency during the lattice Boltzmann and tracer computation required in one growth step.

5.1.5 Comparison of the Load Balancing Strategies In Fig. 14 the load balancing efficiencies (see Eq. 30) are shown for the two load balancing strategies: box decomposition in combination with scattered decomposition, and ORB in combination with scattered decomposition. In this comparison the two strategies were tested for both extreme morphologies of the aggregate: a compact shaped aggregate shown in Fig. 15C and a thin-branching (dendritic shaped) object depicted in Fig. 15A.

5.1.6 Simulated Growth Forms in a Mono Directional Flow Regime In the simulations with a mono directional flow regime it was found that the

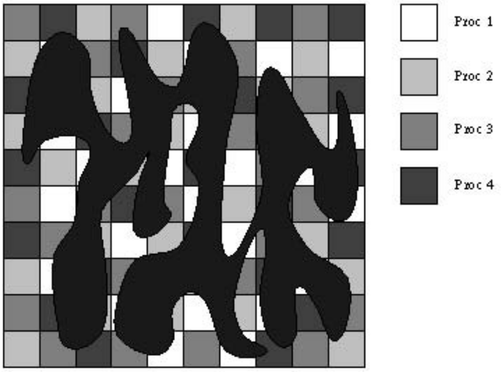


Fig. 13. Decomposition of an irregular shaped object in a 2D lattice. In this case 100 blocks are scattered over 4 processors

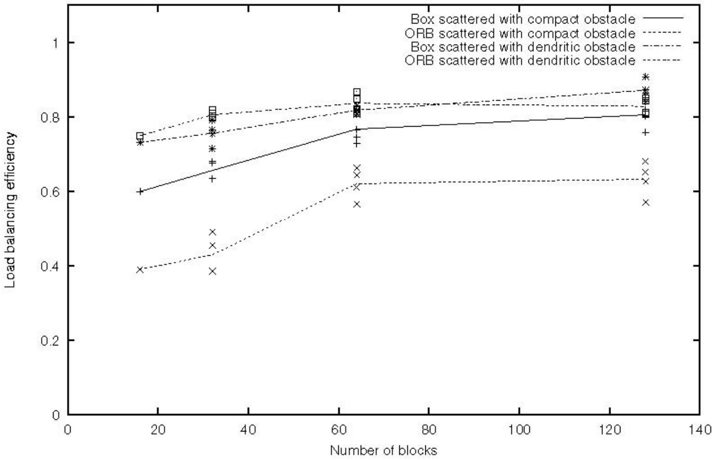


Fig. 14. The load balancing efficiencies of 4 different experiments as a function of the total number of blocks. The dendritic object is shown in Fig. 15A and the compact object is depicted in Fig. 15C

aggregate gradually changes from a thin-branching morphology (diffusion dominates) into a compact shape (flow dominates), for an increasing Péclet number. In Fig. 15 the results of the simulations are summarized by showing slices through the aggregate. The simulation box is sectioned parallel to the direction of the flow. In the sequence A–C in Fig. 15, the Pe number increases. In this picture it can be observed that the degree of compactness increases for larger Pe numbers.

Furthermore, the effect of the mono-directional flow can be clearly observed: the aggregates tend to grow towards the direction of the flow (the flow is directed from the left to the right).

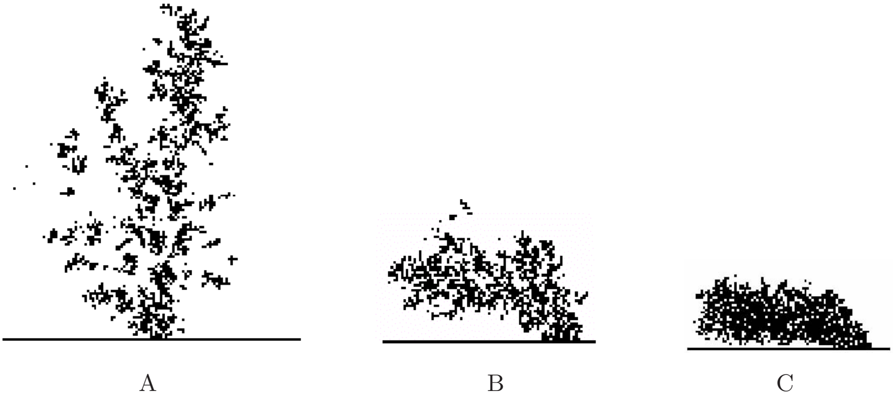


Fig. 15. Slices through the middle of the aggregates in the xy-plane, one-directional flow experiment: A–C, Péclet number increases from approximately 0.0 to 3.0. The flow is directed from the left to the right

5.1.7 Simulated Growth Forms in a Bidirectional (Alternating) Flow Regime In the previous section it is assumed that the growth form develops under mono-directional flow conditions. As a consequence an asymmetric form develops, as shown in Fig. 15, where the aggregates tend to grow in the upstream direction. This trend becomes stronger for higher Pe numbers. In reality, the flow direction will basically reverse twice a day due to the tidal movements. A better approximation is to use a bidirectional flow system by using an aggregation model in which the flow direction is reversed after each growth step [29].

The morphology of the aggregates, in the bidirectional flow experiment, is depicted in Fig. 16.

In Figs. 17 and 18 slices through the simulation box in the xz-plane are shown in which the nutrient distribution is visualized, for respectively the Pe numbers 3.000 (flow dominates) and 0.015 (diffusion dominates). The color shift black–white in these pictures indicate a depletion of nutrients, black indicates the maximum concentration of nutrient, while the regions with nearly zero concentration are shown in white.

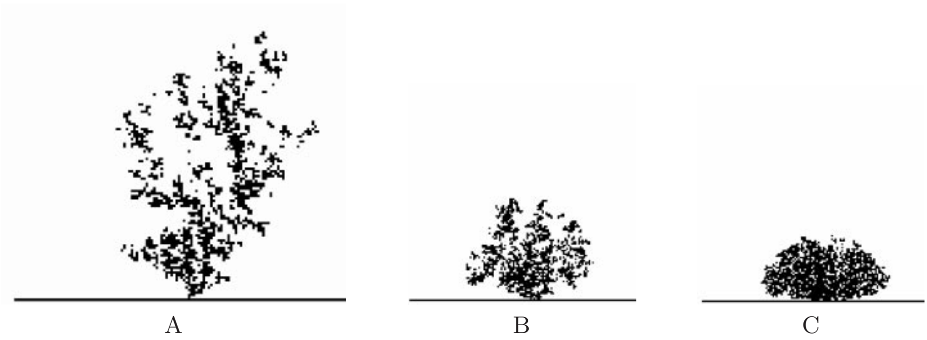


Fig. 16. Slices through the middle of the aggregates in the xy-plane, alternating flow experiment: A–C, Péclet number increases from approximately 0.0 to 3.0

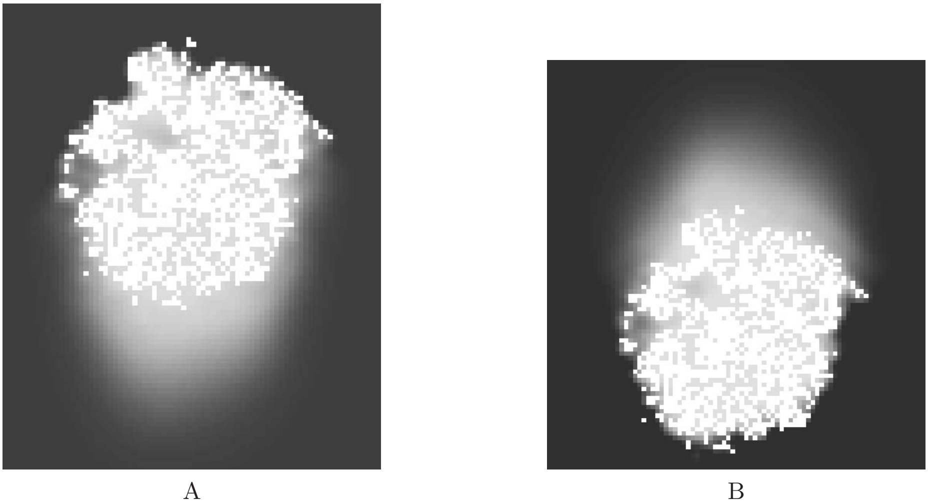


Fig. 17. Slice through the simulation box in the xz-plane showing the nutrient distribution in two successive growth stages in the alternating flow experiment in which Pe is set to the value 3.000 (flow dominates) in the flow is directed from top to bottom in A and directed from bottom to top in B

5.1.8 Discussion

Parallelization aspects When comparing the load balancing efficiencies (Eq. 30) for the compact and dendritic objects in Fig. 14, comparable results are obtained for the dendritic object with both decomposition strategies. For the compact object the best load balancing efficiencies are obtained with the boxed decomposition in combination with scattered decomposition method. The main explanation for the difference, for both strategies, between the compact and the dendritic object is that in the last case the object is already dispersed over space.

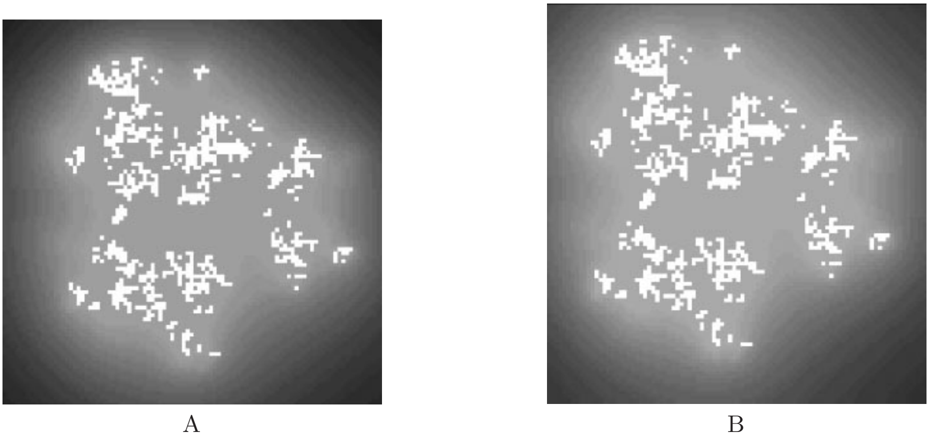


Fig. 18. Slice through the simulation box in the xz-plane showing the nutrient distribution in two successive growth stages in the alternating flow experiment in which Pe is set to the value 0.015 (diffusion dominates) in flow is directed from top to bottom in A and directed from bottom to top in B

This property, the degree of space-filling of the object, can be quantified using the fractal box dimension D_{box} of the object [28]. The fractal dimension D_{box} of the surface of the object can be determined using a 3D version of the (cube)box-counting method described by Feder [18]. In three dimensions its value varies from a minimum of 2, for a solid object with a perfectly smooth surface, to a maximum of 3 for a solid with a space-filling surface. The D_{box} of the compact object was approximately 2.0, while in the dendritic case a value of approximately 2.3 was found. A major disadvantage of both strategies is that, although the load balancing efficiencies increase with the number of blocks used in the scattered decomposition, the communication overhead increases also. The results in Fig. 14 show that the morphology of the object strongly influences the degree of improvement introduced by increasing the number of scattered blocks.

Biological aspects The nutrient distributions shown in Figs. 17 and 18 demonstrate the main differences between diffusion and flow dominated regimes. For low Péclet numbers the distribution of nutrient is roughly symmetric about the center of the aggregate, where the highest concentration reside at the tips of the aggregate and where between the branches an area depleted from nutrients is found with a very low growth probability. At higher Péclet numbers a clear asymmetry develops in the distribution with a depleted region developing downstream of the object (see Fig. 17). As a consequence, there is a low probability of growth in the depleted region. A gradual increase of compactness is demonstrated in Fig. 16 for an increasing influence of hydrodynamics. This gradual increase of compactness corresponds qualitatively to the observations made in stony corals, hydrocorals, and sponges, where growth forms gradually transform from a compact shape, under conditions exposed to water movement, into a thin-branching one under sheltered conditions [25,27,67]. When comparing the slices through the aggregates shown in Figs. 15 and 16 it can also be observed that the increasing degree of asymmetry in the aggregate in the mono-directional flow experiment, for increasing *Pe* numbers, has disappeared in the alternating flow experiment. In the last experiments, aggregates have developed with a roughly radiate symmetry, which corresponds qualitatively to the shape of branching sessile organisms, as for example *Pocillopora damicornis*. These experiments seem to indicate that an alternating flow, a reversal of the flow direction basically twice a day, leads to radiate symmetrical growth forms.

The alternating flow model is a strong simplification of the actual growth process. In many stony corals, as for example the species *Pocillopora damicornis* photosynthesis represents a major energy input. The actual growth process in many sponges, stony-corals, and hydrocorals [25] consists of adding layers of material (varying in thickness) on top of the preceding growth stage, and not by the addition of particles. An accretive growth model, in which layers of material are constructed on top of the previous layers and where the local thickness is determined by the local amount of absorbed nutrients or light intensity [25], also offers possibilities for a quantitative morphological comparison of simulated and actual growth forms [2,26].

5.2 Ising Spin Model Using *Asynchronous* Cellular Automata

The Ising spin model is a model of a system of interacting variables in statistical physics. The model was proposed by Wilhelm Lenz and investigated by his graduate student, Ernst Ising, to study the phase transition from a paramagnet to a ferromagnet [6]. A variant of the Ising spin model that incorporates the time evolution of the physical system is a prototypical example how Asynchronous Cellular Automata can be used to simulate asynchronous temporal behavior. The resulting ACA model is executed using the Time Warp optimistic simulation method (see Section 3).

A key ingredient in the theory of magnetism is the electron's spin and the associated magnetic moment. Ferromagnetism arises when a collection of such spins conspire so that all of their magnetic moments align in the same direction, yielding a total magnetic moment that is macroscopic in size. As we are interested how macroscopic ferromagnetism arises, we need to understand how the microscopic interaction between spins gives rise to this overall alignment. Furthermore, we would like to study how the magnetic properties depend on temperature, as systems generally loose their magnetism at high temperatures.

5.2.1 The Ising Spin Model To introduce the Ising model, consider a lattice containing N sites and assume that each lattice site i has associated with it a number s_i, where $s_i = +1$ for an "up" spin and $s_i = -1$ for a "down" spin. A particular configuration or microstate of the lattice is specified by the set of variables $\{s_1, s_2, \dots, s_N\}$ for all lattice sites (see Fig. 19).

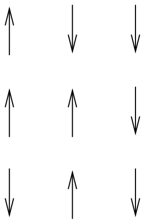


Fig. 19. Schematic spin model for a ferromagnet

The macroscopic properties of a system are determined by the nature of the accessible microstates. Hence, it is necessary to know the dependence of the energy on the configuration of spins. The total energy of the Ising spin model is given by

$$E = -J \sum_{i,j=\mathrm{nn}(i)}^{N} s_i s_j - \mu_0 H \sum_{i=1}^{N} s_i \ , \tag{31}$$

where $s_i = \pm 1$, J is the measure of the strength of the interaction between spins, and the first sum is over all pairs of spins that are nearest neighbors (see Fig. 20). The second term in Eq. 31 is the energy of interaction of the magnetic moment, μ_0, with an external magnetic field, H.

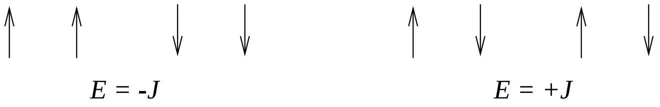


Fig. 20. The interaction energy between nearest neighbor spins in the absence of an external magnetic field

If $J > 0$, then the states ↑↑ and ↓↓ are energetically favored in comparison to the states ↑↓ and ↓↑. Hence for $J > 0$, we expect that the state of the lowest total energy is ferromagnetic, i.e., the spins all point to the same direction. If $J < 0$, the states ↑↓ and ↓↑ are favored and the state of the lowest energy is expected to be paramagnetic, i.e., alternate spins are aligned. If we add a magnetic field to the system, the spins will tend to orient themselves parallel to H, since this lowers the energy.

The average of the physical quantities in the system, such as energy E or magnetization M, can be computed in two ways. The time average of physical quantities are measured over a time interval sufficiently long to allow the system to sample a large number of microstates. Although time average is conceptually simple, it is convenient to formulate statistical averages at a given instant of time. In this interpretation, all realizable system configurations describe an ensemble of identical systems. Then the ensemble average of the mean energy E is given by

$$\langle E \rangle = \sum_{s=1}^{m} E_s P_s \, ,$$

where P_s is the probability to find the system in microstate s, and m is the number of microstates.

Another physical quantity of interest is the magnetization of the system. The total magnetization M for a system of N spins is given by

$$M = \sum_{i=1}^{N} s_i \, .$$

In our study of the Ising spin system, we are interested in the equilibrium quantity $\langle M \rangle$, i.e., the ensemble average of the mean magnetization M.

Besides the mean energy, another thermal quantity of interest is specific heat or heat capacity C. The heat capacity C can be determined by the statistical fluctuation of the total energy in the ensemble:

$$C = \frac{1}{kT^2} \left(\langle E^2 \rangle - \langle E \rangle^2 \right) \, .$$

And in analogy to the heat capacity, the magnetic susceptibility χ is related to the fluctuations of the magnetization:

$$\chi = \frac{1}{kT} \left(\langle M^2 \rangle - \langle M \rangle^2 \right) \, .$$

For the Ising model the dependence of the energy on the spin configuration (Eq. 31) is not sufficient to determine the time-dependent properties of the system. That is, the relation Eq. 31 does not tell us how the system changes from one spin configuration to another, therefore we have to introduce the dynamics separately.

5.2.2 The Dynamics in the Ising Spin Model Physical systems are generally not isolated, but are part of a larger environment. In this respect, the systems exchange energy with their environment. As the system is relatively small compared to the environment, any change in the energy of the smaller system does not have an effect on the temperature of the environment. The environment acts as a heat reservoir or heat bath at a fixed temperature T. From the perspective of the small system under study, it is placed in a heat bath and it reaches thermal equilibrium by exchanging energy with the environment until the system attains the temperature of the bath.

A fundamental result from statistical mechanics is that for a system in equilibrium with a heat bath, the probability of finding the system in a particular microstate is proportional to the Boltzmann distribution [55]

$$P_s \sim e^{-\beta E_s} \, ,$$

where $\beta = 1/k_B T$, k_B is Boltzmann's constant, E_s is the energy of microstate s, and P_s is the probability of finding the system in microstate s.

The Metropolis Algorithm To introduce the dynamics that describes how the system changes from one configuration to another, we need an efficient method to obtain a representative sample of the total number of microstates, while the temperature T of the system is fixed. The determination of the equilibrium quantities is time independent, that is the computation of these quantities does not depend on simulation time. As a result, we can apply Monte Carlo simulation methods to solve the dynamics of the system. The well-known Metropolis algorithm uses the Boltzmann distribution to effectively explore the set of possible configurations at a fixed temperature T [4]. The Metropolis algorithm samples a representative set of microstates by using an importance sampling method to generate microstates according a probability function

$$\pi_s = \frac{e^{-\beta E_s}}{\sum_{s=1}^{m} e^{-\beta E_s}} \, .$$

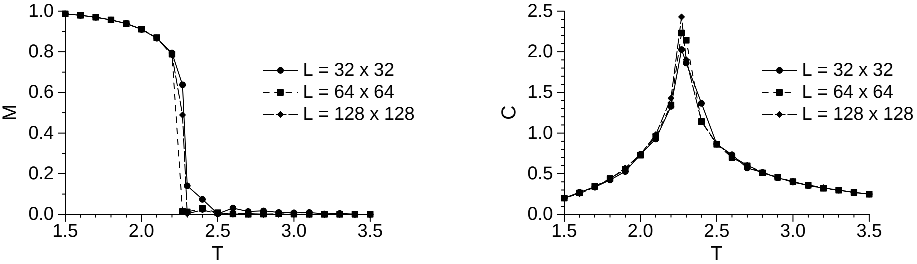
This choice of π_s implies that the ensemble average for the mean energy and mean magnetization can be written as

$$\langle E \rangle = \frac{1}{m} \sum_{s=1}^{m} E_s \quad \text{and} \quad \langle M \rangle = \frac{1}{m} \sum_{s=1}^{m} M_s \, .$$

The resulting Metropolis algorithm samples the microstates according to the Boltzmann probability. First, the algorithm makes a random trial change (a spin flip) in the microstate. Then the energy difference ΔE is computed. The trial is accepted with probability $e^{-\beta \Delta E}$ (note that for $\Delta E \leq 0$ the probability is equal to or larger than one and the trial is always accepted). After the trial, accepted or not accepted, the physical quantities are determined, and the next iteration of the Metropolis algorithm can be started.

The number of Monte Carlo steps per particle plays an important role in Monte Carlo simulations. On the average, the simulation attempts to change the state of each particle once during each Monte Carlo step per particle. We will refer to the number of Monte Carlo steps per particle as the "time," even though this time has no obvious direct relation to physical time. We can view each Monte Carlo time step as one interaction with the heat bath. The effect of this interaction varies according to the temperature T, since T enters through the Boltzmann probability for flipping a spin.

The temperature dependency of the physical quantities $\langle M \rangle$ and C are shown in figures Fig. 21(a) and Fig. 21(b). For temperature $T = 0$, we know that the spins are perfectly aligned in either direction, thus the mean magnetization per spin is ±1. As T increases, we see in Fig. 21(a) that $\langle M \rangle$ decreases continuously until $T = T_c$, at which $\langle M \rangle$ drops to 0. This T_c is known as the critical temperature and separates the ferromagnetic phase $T < T_c$ from the the paramagnetic phase $T > T_c$. The singularity associated with the critical temperature T_c is also apparent in Fig. 21(b). The heat capacity at the transition is related with the large fluctuations found near the critical temperature. The peak becomes sharper for larger systems but does not diverge because the lattice has finite sizes (singularities are only found in an infinite system).



(a) Temperature dependence of the mean magnetization per spin for lattice size 32 × 32, 64 × 64, and 128 × 128

(b) Temperature dependence of the specific heat for lattice size 32 × 32, 64 × 64, and 128 × 128

Fig. 21. Temperature dependency of mean magnetization and specific heat

5.2.3 Continuous-Time Ising Spin System The standard Ising spin model represents a certain discrete-time model, as Monte Carlo steps are regarded to be time steps. However, the transient evolution of the Ising spin configurations is considered an artifact. Glauber [20] introduced continuous-time probabilistic dynamics for the Ising system to represent the time evolution of the physical system.

The Ising spin model with continuous-time probabilistic dynamics cannot be solved by Monte Carlo simulation, since time has no explicit implication on the evolution of the system in the Monte Carlo execution model. To capture the asynchronous continuous-time dynamics correctly, the problem is mapped to the ACA model and is executed by event-driven simulation.

In the continuous-time Ising spin model, a spin is allowed to change the state, a so-called spin flip, at random times. The attempted state change arrivals for a particular spin form a Poisson process. The Poisson arrival processes for different spins are independent, however, the arrival rate is the same for each spin. Similar to the Monte Carlo simulation, the attempted spin flip, or trial, is realized by calculating the energy difference ΔE between the new configuration and the old configuration. The spin flip is accepted with the Boltzmann probability $e^{-\beta \Delta E}$.

The discrete-time and continuous-time models are similar. They have the same distribution of the physical equilibrium quantities and both produce the same random sequences of configurations. The difference between the two models is the time scale at which the configurations are produced: in discrete-time, the time interval between trials is equal, and in continuous-time, the time intervals are random exponentially distributed.

5.2.4 Optimistic Simulation of the Parallel ACA Model The resulting continuous-time Ising spin model is parallelized by geometric decomposition. The Ising spin lattice is partitioned into sub-lattices, and the sub-lattices are mapped onto parallel processors. To minimize the communication between sub-lattices, local copies of the neighbor boundaries are stored locally (see Fig. 22). By maintaining local copies of neighbor boundaries, spin values are only communicated when they are actually changed, rather than when they are only referenced. A spin flip along the boundary is communicated to the neighbors by an event message. The causal order of the event messages, and thus the spin updates, are guaranteed by the optimistic simulation mechanism.

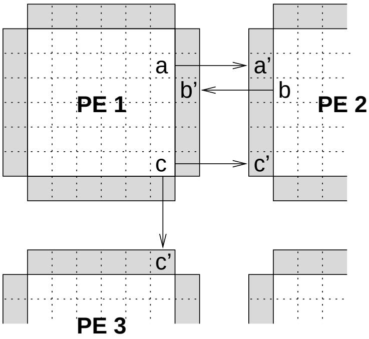


Fig. 22. Spatial decomposition of the Ising spin lattice. The grey areas are local copies of neighbor boundary strips. For example, processor PE 2 has a local copy of spin "a" owned by processor PE 1. Processors PE 2 and PE 3 both own a copy of spin "c". The arrows in the figure indicate the event messages sent upon a spin flip

Asynchronous Cellular Automata, and thus also the Ising spin model, put additional requirements on the original formulation of the Time Warp method. For example, the Time Warp method, as all optimistic PDES methods, must save its state vector each time an event is executed. The state vector of an spatial decomposed ACA can be arbitrarily large, that is, all the cells in the sub-lattice are part of the state vector. For efficient memory management, we incorporate incremental state saving in the Time Warp method [51]. Incremental state saving stores not the full state vector, but saves only the changes to the state vector due to the execution of an event, which is only a small fraction of the full state. Besides efficient memory management, incremental state saving also reduces the time overhead related to the memory copy.

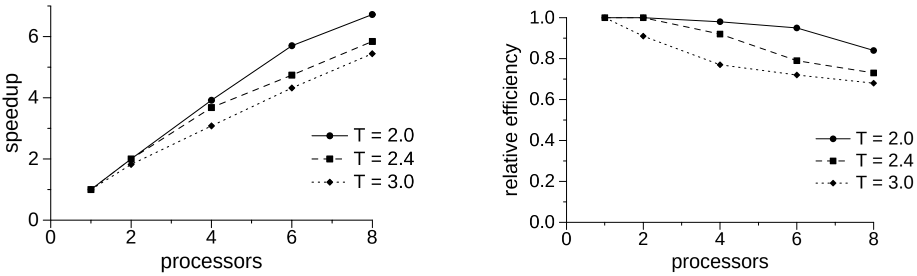
With incremental state saving, no full copy of a state vector at a certain simulation time exists in the simulation execution environment. Instead, upon a rollback of a series of events, the state vector is reconstructed by processing the event–partial state collection in reverse order. Although incremental state saving requires less state saving time and memory, there is an increased cost of state reconstruction. In general, the number of rolled back events is a fraction of the number of events executed during forward simulation. The fraction of rolled back events and the time overhead difference between state saving is an order of 10 bytes versus an order of 10^6 bytes, therefore incremental state saving is favorable in spatial decomposed ACA applications.

Parallel Performance Results To validate the efficacy of the optimistic Time Warp simulation method, we have designed and implemented the continuous-time Ising spin model to study the parallel scalability behavior of the system. The experiments with the Ising spin model were performed on the Distributed ASCI Supercomputer (DAS) [15]. The DAS consists of four wide-area distributed clusters of total 200 Pentium Pro nodes. ATM is used to realize the wide-area interconnection between the clusters, while the Pentium Pro nodes within a cluster are connected with Myrinet system area network technology.

To determine the *speedup* and *relative efficiency* of the parallel Ising spin implementation, the execution time of the parallel simulation on one processor is compared with the execution time on different number of processors. Figure 23(a) shows the relation between speedup and the number of processors for a fixed problem size. Together with the results from Fig. 23(b), we can see that the parallel Ising spin for $T = 2.0$ scales almost linearly up to 6 processors, but eventually drops to a relative efficiency of 0.83 for 8 processors. For temperature $T = 3.0$ the relative efficiency decreases gradually to 0.68 for 8 processors.

The decreasing efficiency is mainly due to the increased costs to synchronize the parallel processes. The difference in parallel performance for different temperatures T can be explained by the measure of dynamic behavior that depends on the temperature of the system. According to the Boltzmann acceptance probability $e^{-\Delta E/k_B T}$, more trails are accepted as the temperature increases. If there are more changes in the system, relatively more synchronization messages must be sent between the sub-lattices, which affects the performance negatively. With the increase of the number of processors, the time overhead to synchronize the parallel simulation processes increases even more as there are more parallel processes that have to find their mutual synchronization point in time.

The parallel ACA model with the Time Warp execution mechanism is an effective solver for continuous-time Ising spin systems. The microscopic ACA rules defining the spin flip probabilities describe the macroscopic magnetization behavior of the Ising spin system (see Fig. 21(a) and Fig. 21(b)). The optimistic simulation method scales reasonably well with the number of parallel processors, although precautions have to be taken. The required overhead time to synchronize the simulation increases with the number of processors. The synchronization overhead can be reduced by limiting the optimism of the Time Warp mechanism. The optimism control effectively bounds the time retardation between the parallel processors such that synchronization between the processors is faster accomplished.



(a) Speedup of parallel Ising spin for lattice size 128 × 128

(b) Relative efficiency of parallel Ising spin for l attice size 128 × 128

Fig. 23. Parallel performance of the Ising spin simulation

6 Summary and Discussion

A common denominator in the next generation scientific computing applications is the understanding of multi-scale phenomena, where systems are studied over large temporal and spatial scales. The simulation of these natural phenomena requires an ever increasing computer performance. Although computer performance as such still doubles approximately every year, it is our strong belief that the development of algorithms for modern computer architectures stays behind. One of the biggest challenges is, therefore, to develop completely new algorithmic approaches that support efficient modeling of natural phenomena and – at the same time – support efficient distributed simulation. Hence we need to boost the *computational* power instead of the *computer* power.

One way to approach this is to look closely to the way nature itself performs computation. This is largely an unexplored research field. In this paper we discussed the concept of interacting virtual particles whose dynamics give rise to complex behavior, by using Cellular Automata as a compute metaphor for modeling and distributed simulation.

As specific instances of Cellular Automata we described the concepts and use of parallel Lattice Gas Automata and the Lattice Boltzmann model. Although these models have been around for a decade, they were mainly studied from a theoretical physics point of view. Our interest is to study them from a computational science point of view, to apply them to real-life natural phenomena and to compare them with real experiments. We are on the front of the second wave of interest in discrete particle models, where the computational aspects and the modeling abilities are the main research questions.

In addition we described a new approach to efficient distributed execution of asynchronous cellular automata through discrete event execution and apply this to different biological and physical models.

In the near future we will setup an international collaboration to use the developed models and concepts in the exploration of various challenging problems stemming from biology, ranging from tumor growth models to population dynamics. Population dynamics models for instance, can be used to understand fluctuations in natural populations, and are fundamental in fishery, ecological research and management of nature reserves. A well known example of a population dynamics model are the Lotka-Volterra equations, first proposed by Volterra to explain the oscillatory levels of certain fish catches in the Adriatic sea.

The inability of Lotka-Volterra models to capture the individual stochastic interaction, has motivated the application of Cellular Automata as an alternative modeling paradigm [45,69]. In the CA model, the populations of preys and predators are no longer considered as homogeneous collections of individuals with identical average properties. CA models form the basis for population dynamics models based on discrete individuals, where the behavior of the individuals is formulated by microscopic rules. An additional advantage is that individual processes, such as movement in space, growth, reproduction, behavioral and ecological interaction, can be represented explicitly.

A synchronous update scheme for the CA model is not realistic from a biological point of view: it is not likely that groups of individuals move simultaneously at the exact same time through space. As each individual behaves independently from the others, both in time and space, an asynchronous update scheme is required. The ACA model and the resulting event-driven simulation associates a simulation time with each update, thus enabling a more meaningful interpretation to the time evolution of individual based models.

Acknowledgments

The authors wish to thank Arjen Schoneveld, David Dubbeldam, and Marco Visser for their contribution and assistance in preparing this paper.

References

1. Y. Bar-Yam. *Dynamics of Complex Systems.* Addison-Wesley, 1997. 213
2. R. G. Belleman, J. A. Kaandorp, and P. M. A. Sloot. A virtual environment for the exploration of diffusion and flow phenomena in complex geometries. *Future Generation Computer Systems*, 14:209–214, 1998. 236
3. H. Bersini and V. Detours. Asynchrony induces stability in cellular automata based models. In *Proceedings of the IVth Conference on Artificial Life*, pages 382–387, Cambridge, MA, July 1994. 214
4. K. Binder and D. W. Heermann. *Monte Carlo Simulation in Statistical Physics.* Springer-Verlag, New York, 1992. 239
5. D. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity.* Prentice-Hall, 1994. 207
6. S. G. Brush. History of the Lenz-Ising model. *Rev. Mod. Phys.*, 39:883, 1967. 236
7. A. W. Burks. *Essays on Cellular Automata.* Univ. Illinois Press, Illinois, 1970. 206

8. F. Celada and P. E. Seiden. A computer model of cellular interactions in the immune system. *Immunology Today*, 13(12):56–62, 1992. 214
9. K. M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, September 1979. 218
10. S. Chen and G. D. Doolen. Lattice Boltzmann method for fluid flows. *Annu. Rev. Fluid Mech.*, 30:329, 1998. 225
11. B. Chopard and M. Droz. *Cellular Automata Modeling of Physical Systems*. Cambridge University Press, 1998. 219, 223
12. J. P. Crutchfield. Critical computation, phase transitions and hierarchical learning. In M. Yamaguti, editor, *Towards the Harnessing of Chaos*, Amsterdam, 1994. Elsevier Science. 212
13. J. P. Crutchfield and M. Mitchell. The evolution of emergent computation. *Proceedings of the National Academy of Sciences*, 92(23):10742, 1995. 209
14. J. P. Crutchfield and K. Young. Inferring statistical complexity. *Phys. Rev. Lett*, 63:105–108, 1989. 212, 212
15. The distributed ASCI supercomputer (DAS). http://www.cs.vu.nl/ bal/das.html. 242
16. D. Dubbeldam, A. G. Hoekstra, and P. M. A. Sloot. Computational aspects of multi-species lattice-gas automata. In P. M. A. Sloot, M. Bubak, A. G. Hoekstra, and L. O. Hertzberger, editors, *Proceedings of the International Conference HPCN Europe '99*, volume 1593 of *Lecture Notes on Computer Science*, pages 339–349, 1999. 226
17. P. A. Dufort and C. J. Lumsden. The complexity and entropy of Turing machines. In *Workshop on Physics and Computation*, Dallas, Texas, 1994. 212
18. J. Feder. *Fractals*. Plenum Press, New York, London, 1988. 235
19. P. Gaspard and X.-J. Wang. Noise, chaos, and (ε, τ)-entropy per unit time. *Physics Letters*, 235(6):291–343, 1993. 210
20. R. J. Glauber. Time-dependent statistics of the Ising model. *Journal of Mathematical Physics*, 4(2):294–307, February 1963. 240
21. P. Grassberger. Long-range effects in an elementary cellular automaton. *J. Stat. Phys.*, 45(1/2):27–39, 1986. 209, 210
22. J. L. Harper, B. R. Rosen, and J. White. *The Growth and Form of Modular Organisms*. The Royal Society London, London, 1986. 227
23. T. E. Ingerson and R. L. Buvel. Structure in asynchronous cellular automata. *Physica D*, 10(1/2):59–68, January 1984. 214
24. D. R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985. 218, 218
25. J. A. Kaandorp. *Fractal Modelling: Growth and Form in Biology*. Springer-Verlag, Berlin, New York, 1994. 236, 236, 236
26. J. A. Kaandorp. Analysis and synthesis of radiate accretive growth in three dimensions. *J. Theor. Biol.*, 175:39–55, 1995. 236
27. J. A. Kaandorp. Morphological analysis of growth forms of branching marine sessile organisms along environmental gradients. *Mar. Biol.*, (in press). 227, 236
28. J. A. Kaandorp, C. Lowe, D. Frenkel, and P. M. A. Sloot. The effect of nutrient diffusion and flow on coral morphology. *Phys. Rev. Lett.*, 77(11):2328–2331, 1996. 227, 227, 229, 235
29. J. A. Kaandorp and P. M. A. Sloot. Growth and form of sponges and corals in a moving fluid. In A. Carbone and M. Gromov, editors, *Pattern Formation in Biology, Dynamics and Computer Graphics*, Singapore. World Scientific. (In press). 227, 227, 233

30. D. Kandhai, A. G. Hoekstra, M. Kataja, J. Timonen, and P. M. A. Sloot. Lattice Boltzmann hydrodynamics on parallel systems. *Comp. Phys. Comm.*, 111:14–26, 1998. 226
31. D. Kandhai, A. Koponen, A. Hoekstra, M. Kataja, J. Timonen, and P. M. A. Sloot. Implementation aspects of 3D lattice-BGK: Boundaries, accuracy and a new fast relaxation technique. In press, J. Comp. Phys., 1999. 226
32. D. Kandhai, D. Vidal, A. Hoekstra, H. Hoefsloot, P. Iedema, and P. Sloot. Lattice-Boltzmann and finite-element simulations of fluid flow in a SMRX static mixer. In press, Int. J. Num. Meth. Fluids, 1999. 226
33. S. A. Kauffman. *The Origins of Order*. Oxford University Press, 1993. 209
34. A. Koponen, D. Kandhai, E. Hellin, M. Alava, A. Hoekstra, M. Kataja, K. Niskanen, P. Sloot, and J. Timonen. Permeability of three-dimensional random fiber webs. *Phys. Rev. Lett.*, 80:716–719, 1998. 226
35. C. G. Langton. Studying artificial life with cellular automata. *Physica D*, 22:120–149, 1986. 209, 209, 210
36. C. G. Langton. Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D*, 42:12–37, 1990. 210
37. W. Li and N. H. Packard. The structure of the elementary cellular automata rule space. *Complex Systems*, 4:281–297, 1990. 208
38. K. Lindgren and M. G. Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4:299–318, 1990. 208
39. B. D. Lubachevsky. Efficient parallel simulation of asynchronous cellular arrays. *Complex Systems*, 1(6):1099–1123, December 1987. 214
40. B. D. Lubachevsky. Efficient parallel simulations of dynamic Ising spin systems. *Journal of Computational Physics*, 75(1):103–122, March 1988. 214
41. E. D. Lumer and G. Nicolis. Synchronous versus asynchronous dynamics in spatially distributed systems. *Physica D*, 71:440–452, 1994. 214
42. J. Machta. The computational complexity of pattern formation. *Journal of Statistical Physics*, 70(3/4):949–967, 1993. 229, 231
43. J. Machta and R. Greenlaw. The parallel complexity of growth models. *Journal of Statistical Physics*, 77:755–781, 1994. 229, 231
44. P. Manneville, N. Boccara, G. Y. Vichniac, and R. Bidaux, editors. *Cellular Automata and Modeling of Complex Physical Systems*, volume 46 of *Springer Proceedings in Physics*. Springer-Verlag, 1989. 206
45. E. McCauley, W. G. Wilson, and A. M. de Roos. Dynamics of age-structured and spatially structured predator-prey interactions: Individual based models and population-level formulations. *The American Naturalist*, 142(3):412–442, 1993. 244
46. M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, N.J., 1967. 212
47. M. Mitchell. Computation in cellular automata: A selected review. In T. Gramss, S. Bornholdt, M. Gross, M. Mitchell, and T. Pellizzari, editors, *Non-Standard Computation*. Wiley-VCH, 1998. 212
48. M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Dynamics, computation, and the ‘edge of chaos’: A re-examination. In G. Cowan, D. Pines, and D. Melzner, editors, *Complexity: Metaphors, Models, and Reality*, 1994. 212
49. C. Moore and M. G. Nordhal. Lattice gas prediction is p-complete. Technical report, Santa Fe Instute for Complex studies, 1997. SFI 97-04-043. 220
50. D. M. Nicol and J. H. Saltz. An analysis of scatter decomposition. *IEEE transactions on computers*, 39(11):1337–1345, 1990. 231

51. B. J. Overeinder and P. M. A. Sloot. Application of Time Warp to parallel simulations with asynchronous cellular automata. In *Proceedings of the 1993 European Simulation Symposium*, pages 397–402, Delft, The Netherlands, October 1993. 214, 241
52. B. J. Overeinder and P. M. A. Sloot. Breaking the curse of dynamics by task migration: Pilot experiments in the polder metacomputer. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 1332 of *Lecture Notes in Computer Science*, pages 194–207, Berlin, 1997. Springer-Verlag. 204
53. B. J. Overeinder, P. M. A. Sloot, and L. O. Hertzberger. Time Warp on a Transputer platform: Pilot study with asynchronous cellular automata. In *Parallel Computing and Transputer Applications*, pages 1303–1312, Barcelona, Spain, September 1992. 214
54. N. H. Packard. Adaptation toward the edge of chaos. In J.A.S. Kelso, A.J. Mandell, and M.F. Shlesinger, editors, *Dynamic Patterns in Complex Systems*, 1988. 212
55. F. Reif. *Fundamentals of Statistical and Thermal Physics*. McGraw-Hill, New York, 1965. 238
56. J. F. de Ronde. *Mapping in High Performance Computing*. PhD thesis, Department of Computer Science, University of Amsterdam, Amsterdam, The Netherlands, February 1998. 205
57. J. F. de Ronde, A. Schoneveld, and P. M. A. Sloot. Load balancing by redundant decomposition and mapping. In H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, editors, *High Performance Computing and Networking (HPCN'96)*, pages 555–561, 1996. 231, 231
58. D. H. Rothman and S. Zaleski. *Lattice-Gas Cellular Automata, Simple Models of Complex Hydrodynamics*. Cambridge University Press, 1997. 219, 223, 224, 225
59. A. Schoneveld, J. F. de Ronde, and P. M. A. Sloot. Task allocation by parallel evolutionary computing. *Journal of Parallel and Distributed Computing*, 47(1):91–97, 1997. 205
60. W. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois, Urbana, 1949. 209
61. H. D. Simon. Partioning of unstructured problems for parallel processing. *Computing Systems in Engeneering*, 2(2/3):135–148, 1991. 231
62. P. M. A. Sloot. High performance simulation. *EUROSIM – Simulation News Europe*, (15):15–18, 1995. 203
63. P. M. A. Sloot. Modelling for parallel simulation: Possibilities and pitfalls, invited lecture. In *Eurosim'95, Simulation congress*, pages 29–44, Amsterdam, the Netherlands, 1995. 203
64. P. M. A. Sloot, A. Schoneveld, J. F. de Ronde, and J. A. Kaandorp. Large scale simulations of complex systems Part I: Conceptual framework. Technical report, Santa Fe Instute for Complex studies, 1997. SFI Working Paper: 97-07-070. 204
65. P. M. A. Sloot and D. Talia. Parallel cellular automata: Special issue on cellular automata. *Future Generation Computer Systems*, 1999. (In press). 207
66. S. Ulam. Some mathematical problems connected with patterns of growth figures. In A.W. Burks, editor, *Essays on Cellular Automata*, Illinois, 1970. Univ. Illinois Press. 206
67. J. E. N. Veron and M. Pichon. *Scleractinia of Eastern Australia Part 1*, volume 1 of *Australian Institute of Marine Science Monograph Series*. Australian Government Publishing Service, Canberra, 1976. 227, 236
68. J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois, Urbana, 1966. 206

69. W. G. Wilson, A. M. de Roos, and E. McCauley. Spatial instabilities within the diffusive Lotka-Volterra system: Individual-based simulation results. *Theoretical Population Biology*, 43:91–127, 1993. 244
70. A. T. Winfree, E. M. Winfree, and H. Seifert. Organizing centers in cellular excitable medium. *Physica D*, 17:109, 1985. 213
71. T. A. Witten and L. M. Sander. Diffusion-limited aggregation, a kinetic critical phenomenon. *Phys. Rev. Lett.*, 47(19):1400–1403, 1981. 229
72. S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984. 208, 208, 209
73. S. Wolfram. *Cellular Automata and Complexity*. Addison-Wesley, 1994. 206, 208
74. B. P. Zeigler. *Theory of Modelling and Simulation*. John Wiley and Sons, Inc., New York, 1976. 214
75. Bernard P. Zeigler. Discrete event models for cell space simulation. *International Journal of Theoretical Physics*, 21(6/7):573–588, 1982. 214

Supporting Group-By and Pipelining in Bitmap-Enabled Query Processors

Alejandro P. Buchmann and Ming-Chuan Wu

Database Research Group, Computer Science Department
Technische Universität Darmstadt, Germany
Tel: ++49-6151/16 6236   Fax: ++49-6151/16 6229
{wu,buchmann}@dvs1.informatik.tu-darmstadt.de

Abstract. The Group-By operation is widely used in relational Data Warehouses for the aggregation and presentation of results according to user-defined criteria. Rearranging Group-By and join operations has been proposed as an optimization technique that reduces the size of the input relation. Pipelining is another optimization technique that promotes intra-query parallelism by transfering intermediate results to the next operation without materializing them. In Data Warehouses and environments in which exploratory on-line queries are common, pipelining can be used both for optimizing joins and for the reduction of response time by presenting partial results to the user. Efficient pipelining often depends on the order of the intermediate results. Order-dependent operations, such as Group-By, typically require the complete result set before ordering it, thereby making efficient pipelining impossible. In this paper we exploit bitmap indexing to implement Group-By in a manner that can be used for pipelining. Algorithms are presented for different assumptions about buffer availability and for different sort orders.

1 Introduction

Data Warehouses integrate the data from on-line transaction processing systems and a variety of other internal and external sources to provide historical data that can be used in support of the decision making process. Data Warehouses must be capable of responding, in addition to predefined report-type queries, to interactive exploratory queries. In the interactive mode the perceived response time, i.e. the time between issuing a query and the appearence of the first results, is critical.

The size of typical Data Warehouses, their peculiar design in form of variants of the star-schema, and the fact that data access is primarily for reading purposes have an impact on the storage structures that are used, the indexing methods or secondary access paths, the implementation of the operators of the relational algebra and the query processing strategies followed.

The design of a Data Warehouse is often based on a star schema. In this design the aggregatable data from a business transaction, for example, the number of

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 249–260, 1999.

units sold and the value of the sale, are placed in a fact table that is linked through a foreign key relationship with the dimension tables that contain all the descriptive information. As a rule of thumb, fact tables contain many small records but the historical nature of a Data Warehouse causes the fact tables to be extremely large. The dimension tables typically consist of relatively few but long records with many descriptive fields. Dimension tables are in general disjoint and have no attributes in common.

The disjoint nature of the dimension tables has an important effect on the processing of joins: any pairwise join involves the large fact table. To be efficient, join algorithms must be developed that minimize the access to the fact table, ideally reducing it to a single scan or even less through the use of precomputed join indexes, or by exploiting clustering and the corresponding metadata. In addition, the writing of intermediate results, which may in turn be rather large, should be avoided whenever possible. This fact and the need to present (partial) results to the user or the next processing step as soon as possible make pipelined joins desirable.

Another key operation in Data Warehousing is the Group-By. The Group-By operation applies one of the standard SQL aggregation operators (SUM, COUNT, MAX, MIN, AVG) to an aggregatable attribute and groups the result according to the specified grouping attribute. For example, car sales can be summed up and grouped by model. Group-Bys are often applied as the last operation before the ordered results are presented to the user. Previous research has shown the advantage of pushing Group-Bys past joins to reduce the size of the input relations [9,3]. We contend that Group-Bys are prime candidates for pipelining, thus allowing either the user or the next operation to use partial results as they become available.

In [5] sampling is used to produce approximate results for online aggregation. While a statistical approach with gradual refinement may be enough to satisfy user requirements in some decision support environments, it is not adequate as input to other operations. In this paper we show how bitmap indexes can be exploited to produce online *exact* results of the Group-By operation. These results are produced in the proper sort order and can be used both to reduce the turnaround time and to optimize the execution time through pipelining.

Bitmap indexes are popular secondary access methods for Data Warehouses [6,1,7,2,8]. Bitmaps are compact data structures that have a low processing cost and are easy to maintain. In contrast to id-list indexes, such as B-trees, the information of key values and their tuple-ids are encapsulated in bits and bit positions. The operations on bitmaps are inexpensive bitwise operations. Bitmaps offer the possibility of evaluating selection predicates directly on the index and of combining different selection criteria through logical operations on the index without having to access the underlying data. Using id-list indexes, compound selection predicates can either be evaluated by choosing the most selective index to filter the operand table followed by a sequential scan on the temporary table and the evaluation of the rest of the predicates, or directly by a multiple index scan, which uses all the available indexes to retrieve sets of tuple-ids, followed by set operations on them. Both of these methods are more expensive than the use of bitmaps.

Query processing strategies depend heavily on the available access paths and on the implementation of the operators. Since the availability of bitmaps greatly impacts the query processing strategies, the query processor should be bitmap-enabled, for example, by processing queries on the index without retrieving the actual data. One crucial advantage derived from the use of bitmaps is the fact that the intermediate results derived from index processing are themselves bitmap indexes that can be further exploited. So far, efforts reported in the literature have concentrated on the development of new variants of bitmaps and their use and effectiveness in processing individual operations, mostly selections. To take full advantage of the new index structures and the algorithms that use them for individual operations, query processors/optimizers must be bitmap-enabled to exploit the new indexes for the global query optimization process. They should also include optimization criteria that favor pipelining and the early presentation of results to the user thus reducing the perceived response time.

The remainder of this paper briefly reviews bitmap indexing in Section 2 and pipelining in Section 3; Section 4 discusses the optimization of Group-Bys; Section 5 shows how the Group-By operation can be supported through the use of bitmap indexes; and Section 6 discusses future work and provides conclusions.

2 Bitmap Indexing Revisited

Bitmap indexes of different structure have been proposed as secondary access methods for Data Warehouses [6,1,7,2,8]. In principle, bitmap indexes encapsulate the information of attribute values and tuple-ids in bits and bit positions. In their simplest form, bitmap indexes store a bit vector for each value of the domain of an attribute. The length of a bit vector is equal to the cardinality of the indexed table. A “1” is stored in the vector at the position that corresponds to the position of the tuple if the attribute of that tuple has the value of the corresponding bit vector. Otherwise a “0” is stored. Simple bitmap indexing works well for small domains but the vectors become rather sparse in large domains. To solve this problem the bit vectors can either be compressed, resulting in a large number of short and dense vectors, or they can be *encoded*, resulting in a small (logarithmic) number of regular length dense vectors [7]. Encoded Bitmap Indexing preserves not only the positioning of the bits but also the simplicity of application.

An encoded bitmap index defines first an encoding function that is reflected in a mapping table and then defines the corresponding Retrieval Boolean Function for each attribute value. For example, given an attribute A with the domain $\{a, b, c, d, e, f, t, u, v, w\}$, an Encoded Bitmap Index on A consists of a *mapping table*, a set of *Boolean retrieval functions* and a set of bitmaps, as shown in Figure 1.

A	$\mathbf{b}_3$	$\mathbf{b}_2$	$\mathbf{b}_1$	$\mathbf{b}_0$
a	0	0	1	0
c	0	1	1	0
f	0	1	0	1
t	1	1	1	1
v	1	1	0	1

Keys	Encoding	Keys	Encoding
void	$0000_{(2)}$	e	$0100_{(2)}$
NULL	$0001_{(2)}$	f	$0101_{(2)}$
a	$0010_{(2)}$	t	$1111_{(2)}$
b	$0011_{(2)}$	u	$1110_{(2)}$
c	$0110_{(2)}$	v	$1101_{(2)}$
d	$0111_{(2)}$	w	$1100_{(2)}$

(a) Mapping table

$$\begin{array}{lll}
f_{void} = \bar{\mathbf{b}}_3\bar{\mathbf{b}}_2\bar{\mathbf{b}}_1\bar{\mathbf{b}}_0 & f_c = \bar{\mathbf{b}}_3\mathbf{b}_2\mathbf{b}_1\bar{\mathbf{b}}_0 & f_t = \mathbf{b}_3\mathbf{b}_2\mathbf{b}_1\mathbf{b}_0 \\
f_{\mathsf{NULL}} = \bar{\mathbf{b}}_3\bar{\mathbf{b}}_2\bar{\mathbf{b}}_1\mathbf{b}_0 & f_d = \bar{\mathbf{b}}_3\mathbf{b}_2\mathbf{b}_1\mathbf{b}_0 & f_u = \mathbf{b}_3\mathbf{b}_2\mathbf{b}_1\bar{\mathbf{b}}_0 \\
f_a = \bar{\mathbf{b}}_3\bar{\mathbf{b}}_2\mathbf{b}_1\bar{\mathbf{b}}_0 & f_e = \bar{\mathbf{b}}_3\mathbf{b}_2\bar{\mathbf{b}}_1\bar{\mathbf{b}}_0 & f_v = \mathbf{b}_3\mathbf{b}_2\bar{\mathbf{b}}_1\mathbf{b}_0 \\
f_b = \bar{\mathbf{b}}_3\bar{\mathbf{b}}_2\mathbf{b}_1\mathbf{b}_0 & f_f = \bar{\mathbf{b}}_3\mathbf{b}_2\bar{\mathbf{b}}_1\mathbf{b}_0 & f_w = \mathbf{b}_3\mathbf{b}_2\bar{\mathbf{b}}_1\bar{\mathbf{b}}_0
\end{array}$$

(b) Retrieval min-terms

Fig. 1. An Example of Encoded Bitmap Indexing on A

The bitmaps, $\mathbf{b}_3, \mathbf{b}_2, \mathbf{b}_1, \mathbf{b}_0$, are set according to the encoded values of the indexed attribute, *e.g.*, for those tuples with $A = a$, their corresponding bits in $\mathbf{b}_3, \mathbf{b}_2, \mathbf{b}_0$ are cleared, and the bits in $\mathbf{b}_1$ are set to 1. The retrieval Boolean functions are also defined based on the encoded values, *e.g.*, the retrieval Boolean function for the value "a", denoted by f_a, is defined by $\bar{\mathbf{b}}_3\bar{\mathbf{b}}_2\mathbf{b}_1\bar{\mathbf{b}}_0$, where a '0' in the encoded value is expressed by the negation of the Boolean variable. Using the Encoded Bitmap Index on A to evaluate $A \in \{a, b, c, d\}$, the retrieval Boolean functions of the selected values are selected and composed to form a logical disjunction, *i.e.*, $f_a + f_b + f_c + f_d$, which is further reduced to $\bar{\mathbf{b}}_3\mathbf{b}_1$. The 1's bits in the resulting bitmaps indicate those tuples satisfying the selection condition.

There are other variations of bitmap indexing, such as bit-sliced indexes [6,1,2]. In spite of many variations of bitmaps, there are two points in common: 1) the design principle is to reduce the space requirement of the indexes while preserving the performance; and 2) the indexes are *complete*, *i.e.*, for selections of any given value a single bitmap can be generated, and those and only those "1" bits indicate the desired tuples. With these properties, our methods discussed later can be applied to other variation of bitmap indexes, not just Encoded Bitmap Indexes.

The bitmaps we have described so far are all *tuple-level* bitmaps. That means, one bit is used to represent one tuple in the base table. In the processing of Group-Bys and other operations, such as Joins, *page-level* bitmaps are used, because of the nature of block I/Os. One tuple-level bitmap is uniquely mapped to one page-level bitmap. The page-level bitmap is constructed as follows. Suppose that we have a bitmap $\mathbf{b}$, where $|\mathbf{b}|$ denotes the length of $\mathbf{b}$ in bits, and

w is the *blocking factor* of a physical page, *i.e.*, the number of tuples in a page. Then, the page-level bitmap of $\mathbf{b}$, denoted by $\mathbf{b}_p$, consists of $\lceil \frac{|\mathbf{b}|}{w} \rceil$ bits. The i-th bit in $\mathbf{b}_p$ is set, if any bit between the $(i \cdot w)$-th and the $((i+1) \cdot w - 1)$-th bits in $\mathbf{b}$ is set, where $0 \leq i < \lceil \frac{|\mathbf{b}|}{w} \rceil$. In other words, the i-th bit in $\mathbf{b}_p$ is set, if any tuple in the i-th page is selected based on $\mathbf{b}$.

3 Pipelining

When the cost of an operation is evaluated, a significant portion of that cost is due to the materialization and the necessary writing of the result, particularly in situations in which the materialized intermediate result cannot be kept in main memory and must be written to disk. A convenient way to reduce this cost is through pipelining.

When multiple operations must be performed in sequence it is more efficient to pass on the intermediate result to the next operation directly, thereby avoiding the cost of writing for the first operation and the cost of reading for the next operation. For example, if a join of three tables T1, T2, T3 must be performed, it can be performed by joining first T1 with T2 and then joining the resulting table T4 with T3. Instead of waiting until the whole join is performed and writing the intermediate result, the query processor can initiate the second join with the tuples of the result from the first join as these become available. If the results are produced in the right order, an inexpensive merge join can be performed. However, if the intermediate result is not in the right sort order a more expensive join implementation must be chosen or the intermediate result must first be materialized and sorted.

In Data Warehousing the processing of the fact tables must be optimized. Fortunately, those joins are between the partial keys of the records in the fact table which are the keys of the dimensions. The load process of the Data Warehouse typically produces good clustering that can be exploited for pipelining. For example, data corresponding to all the transactions of the day at a given store are loaded overnight, and result in a clustering by date and store. A coarse page level bitmap index or a hierarchically encoded bitmap index can reflect these facts and can be used to support pipelined joins. Pipelining can improve intra-query parallelism drastically in the typical multiprocessor environments.

In environments in which exploratory queries are common, a user often issues a query online and waits for its result. Pipelining has two advantages in such an environment: a) the response time is reduced for the user, since partial results can be displayed as they become available, and b) substantial savings can be realized if the user decides to terminate the query after inspecting the first results.

However, the user typically expects the data to be grouped and sorted by some criterion before it is presented. When the sort order that is expected by the user does not correspond to the order in which the results are generated, the result must first be completely computed and later sorted. Therefore, implementations of the Group-By operation that produce the results incrementally in the right order and can be pipelined, offer a large potential for query optimization.

4 Optimization of Group-By

A generic Data Warehouse query has the following form:

SQL 1

```
Select    non-aggregate attributes
          aggregation(aggregatable attribute)
From      dimension tables
          fact table
Where     join conditions
          restriction conditions
Group-By  non-aggregate attributes
Order-By  sort attributes
```

Conventional optimizers are likely to produce query trees in which the restriction conditions are applied first, followed by the Joins, executing the Group-Bys after the joins, followed by sorting for the final presentation order.

We will illustrate a possible optimization of the Group-By statement with a small example taken from [3].

Example 1. Given is the Data Warehouse of a factory, that models the orders of its dealers in a fact table Order and three dimension tables, Division, Product and Dealer. The factory has some sectors. A sector consists of some divisions, and a division produces some products. The schema is as follows.

```
Division(DivID, SectorID)
Product(ProdID, Cost, DivID)
Order(OrderID, ProdID, DealerID, Amount, Date)
Dealer(DealerID, State, Address)
```

Note that this schema has been normalized. In the terminology of Data Warehousing this normalized schema results in a snowflake, i.e. the dimensional attributes that form a hierarchy must be combined again through join operations on the subdimensions. Therefore, in Data Warehousing it is often recommended that attributes that are functionally dependent on other dimensional attributes are placed in the same dimension table. In this case, SectorID could have been placed in the Product dimension since redundance and updates are not a major issue in a Data Warehouse. We will illustrate the optimization first on the normalized (snowflake) example and will later show that the same optimization is valid for the common (non-normalized) Star schema design.

Suppose that we want to query the sum of the orders for each sector of the factory. In SQL this query would be:

SQL 2

Select	*sum*(Amount)
From	Order, Product, Division
Where	Order.ProdID = Product.ProdID And
	Product.DivID = Division.DivID
Group-By	Division.SectorID

The query trees of SQL 2 are depicted in Figure 2. Figure 2(a) shows the execution tree generated by traditional query optimizers which postpone Group-Bys until all joins are done. In [9,3], it is proposed to process Group-Bys before joins. The benefit of pushing down the Group-Bys in the execution tree is the reduction of the intermediate result which is the input to the join and in turn reduces the join cost. Figure 2(b) shows an execution tree resulting from pushing down the Group-Bys. In the query tree of Figure 2(b) the first Group-By is performed on the base data, while subsequent Group-Bys are performed on intermediate results. Since operations that are performed on the base data usually can exploit existing index structures while intermediate results are not indexed, there is a clear advantage to performing the operations on base data.

Figure 2(c) depicts the execution tree of SQL 2 by pushing all the Group-Bys in Figure 2(b) down to the lowest operator level.

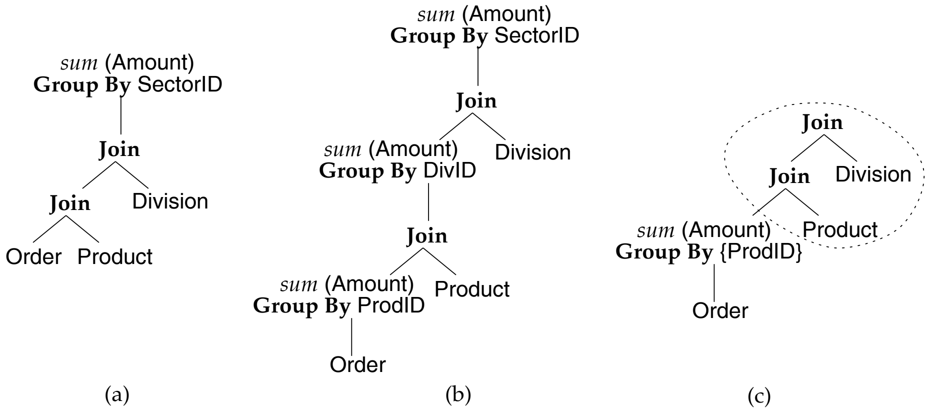


Fig. 2. Execution trees of SQL 2

Pushing the Group-Bys to the level of the base data is possible whenever functional dependencies exist that uniquely determine the grouping attributes at the higher level. This situation will always arise when snowflaking was used to break out hierarchies in a dimensional attribute. For example, ProdID functionally determines DivID (ProdID → DivID), which in turn functionally determines SectorID (DivID → SectorID). The Group-By can be pushed down if no other attributes of the snowflaked dimension(s) are needed in the final result. In the example, the

Group-By can be executed on the corresponding *subsets* of ProdID since there exists one partitioning on the attribute domain of ProdID, such that it divides its attribute domain into several disjoint subsets, and all the values in one subset map to the same value of SectorID.

In the more common Star-schema functionally dependent attribute hierarchies often are placed in the same dimension table as the dimension table's key. The dimension table's primary key is used as part of the fact table's key. As long as functional dependencies exist between the dimension table's primary key and the grouping attribute specified in the Group-By clause, it is possible to find partitions of primary keys that correspond directly to the higher-level grouping attribute, and the Group-By can then be executed directly on the base data.

If the functional dependencies among attributes are modelled in the data catalog, then they can be directly applied to find the partitioning, in our example the partitioning on ProdID. Or, if an Encoded Bitmap Index (EBI) is defined on the products and their hierarchies, namely "sectors–divisions–products", the information about the dimension hierarchies can be hashed into the mapping table together with ProdIDs.

5 Execution of Group-By

The interesting question is whether the Group-By can be executed on the base tuples in such a way, that the result is produced in the required order that can be used for pipelining. Traditionally, if the result of the Group-By operation can be fit into the main memory, the operation can be evaluated by sequentially scanning the operand table once and sorting the result of the aggregation acording to the grouping attribute. If the result of the Group-By does not fit in memory, hashing or sorting are first applied to the operand table, followed by some merges of the intermediate results to produce the final result [4]. These implementations try to optimize query execution time by minimizing the execution time of individual operations.

In Data Warehousing, however, response time is often a more important performance metric than query processing time. Under this circumstance, the key question is: *"how to reschedule the page accesses such that we can minimize the turn-around time, and at the same time keep the query processing cost low?"*

In this section we present Group-By algorithms that exploit bitmap indexing. With the help of bitmaps, we are able to reschedule the I/O accesses to produce Group-By results that can be pipelined and can be used to minimize the turn-around time, which is defined by the time interval from query submission until the first result is available.

The first algorithm, for which the pseudocode is shown in Figure 3, requires sufficient buffer space to buffer all the intermediate results of the Group-By aggregation as well as the pertinent index structures.

We assume that a bitmap index exists on the fact table, more precisely on the partial key that is the primary key of the dimension table in which the grouping attribute is located. Based on functional dependencies, the mapping

Algorithm 1 [Online Grouping by Bitmaps]

```
Input:   grouping attribute(s) $G$, whose domain is further divided into
         a sequence of grouping subsets, $\langle g_1, \ldots, g_k \rangle$, $g_i \subseteq domain(G)$
         and $\{b_1, \ldots, b_k\}$ are the grouping bitmaps for $\{g_1, \ldots, g_k\}$, respectively
         an aggregate function, $f$, and the attribute to be aggregated, $A$
         an operand table, $T$
Output:  grouping results

Begin
  define a bit vector $toDo$ with the same length as $b_i$, $(1 \leq i \leq k)$;
  let $toDo = \mathbf{1}$;
  define a perfect hashing function, $ht$,
                        such that $ht(v) = ht(v'), \forall v, v' \in g_i \ (1 \leq i \leq k)$;
  define a hash table, $HT$, $HT[i]$ denotes the $i$-th entry of $HT$
  for each $b_i$ in the sequence $\langle b_1, \ldots, b_k \rangle$
      for each $j$-th bit in $b_i$, denoted by $b_i[j]$
          if ($b_i[j]$ & $toDo[j]$)
              read in the $j$-th page of $T$, denoted by $T[j]$;
              for each tuple $t$ in $T[j]$
                  cumulate $t.A$ into $HT[ht(t.G)]$;
              clear $toDo[j]$;
      output $f(HT[ht(g)])$, for any $g \in g_i$;
End
```

Fig. 3. Bitmap-based Group-By, unlimited buffer

from the grouping attribute to sets of the dimension key (a partial key of the fact table) is performed. For the purpose of this algorithm we do not require a particular type of bitmap index, since a page-level bitmap $\{b_1, \ldots, b_k\}$ for the set of grouping subsets $\langle g_1, \ldots, g_k \rangle$ can be derived, i.e. for each value in the domain of the grouping attribute a bit vector can be maintained in which a 1 is set if the page of the fact table contains a tuple that is relevant for the aggregation under that particular value of the grouping attribute. The Group-By operation is then performed in the proper sort order. In the first aggregation, with the help of the page level bitmap all pages are read and scanned that are needed for the first aggregation subset g_1. The first aggregation is complete after this pass and can be pipelined. The partial aggregations corresponding to other aggregation subsets are kept in memory but the pages that have been processed are discarded. For each page that has been processed, the corresponding position is set to zero in the ToDo vector, an auxiliary bit vector of the same length as the page-level bitmaps. For the second aggregation subset the corresponding page-level bit vector is ANDed with the ToDo vector and the resulting pages are retrieved and processed. This time the aggregation corresponding to the second aggregation subset will be completed and can be pipelined. After updating the ToDo vector the process is repeated until all subsets are processed. Note that no pages are read twice and they can be discarded right away.

If the buffer is limited and all the values for the incomplete aggregations cannot be kept in main memory, the above algorithm must be modified. Figure 4 presents the pseudocode of the modified algorithm for the case of limited buffer space. In this variant of the algorithm, only a portion of the hash table containing the partially aggregated values and a limited number of bitvectors can be maintained in memory. Therefore, the aggregations are processed by subsets of g. Bitvectors that correspond to the already processed values of g are cleared and replaced by the next group. Since not all aggregations can be performed at the same time, some tuples may have to be read more than once, i.e. some pages may have to be accessed repeatedly. This algorithm is suboptimal if the Group-By is considered in isolation but allows pipelining by producing the results incrementally in the right order of the grouping attribute.

Algorithm 2 [Online Grouping by Bitmaps with Limited Buffer Size]

Input: grouping attribute(s) G, whose domain is further divided into
a sequence of grouping subsets, $\langle g_1, \ldots, g_k \rangle$, $g_i \subseteq \mathit{domain}(G)$
and $\{b_1, \ldots, b_k\}$ are the grouping bitmaps for $\{g_1, \ldots, g_k\}$, respectively
an aggregate function, f, and the attribute to be aggregated, A
an operand table, T
a buffer of size M
Output: grouping results

```
Begin
    define a hash table, $HT$, $HT[i]$ denotes the $i$-th entry of $HT$, such that $i \leq M$;
    define a perfect hashing function, $ht$, such that $ht(v) = ht(v')$, $\forall v, v' \in g_{(i\%M)+1}$;
    let $m = 1$, $n = M$;
    for each $b_i$ in the sequence $\langle g_1, \ldots, g_k \rangle$
        for each $j$-th bit in $b_i$, denoted by $b_i[j]$
            if ($b_i[j]$)
                read in the $j$-th page of $T$, denoted by $T[j]$;
                for each $t$ in $T[j]$
                    cumulate $t.A$ into $HT[ht(t.G)]$, if $t.A \in g_l$, $m \leq l \leq n$;
                clear all $b_l[j]$, $\forall m \leq l \leq n$;
        output $f(HT[ht(g)])$, for any $g \in g_i$;
        let $m = m + 1$, $n = n + 1$;
End
```

Fig. 4. Bitmap-based Group-By, limited buffer

So far we have considered only the common case where the resulting sort order is determined by the grouping attribute. However, from the generic query we saw that other sort orders may be specified through a separate Sort By clause. In particular we are interested in the case in which the sort order is based on the aggregated attribute, for example, the user wants to aggregate the sales of cars grouped by model but needs them sorted by total sales. By providing implementations that combine several operations, it becomes possible to optimize the execution without having to modify the language semantics. Figure 5 shows the pseudocode of an algorithm that produces the result of a Group-By in the order of the aggregation attribute. Through the use of bitmap indexing and the auxiliary information that is automatically available in such an index, i.e. the number of tuples in each group, it becomes possible to produce the aggregations in the proper sort order, thus making it possible to pipeline partial results.

Algorithm 3 [Online Ordering by Bitmaps]

Input: grouping attribute(s) G, whose domain is further divided into
a sequence of grouping subsets, $\langle g_1, \dots, g_k \rangle$, $g_i \subseteq \mathit{domain}(G)$ and
$\{b_1, \dots, b_k\}$ are the tuple-level grouping bitmaps for $\{g_1, \dots, g_k\}$, respectively
an aggregate function, f, and the attribute to be aggregated, A
numbers of tuples in each group, $c_1, \dots, c_k$
an operand table, T, and
vertical bitwise partition on A, denoted by $A_{m-1}, \dots, A_0$
Output:grouping results

```
Begin
  define arrays $sum[]$ and $seq[]$ of $k$ integers;
  let $sum[i] = b_i^{\mathsf{T}} \cdot A_{m-1}$, $i = 1, \dots, k$; // $b_i^{\mathsf{T}}$ denotes the transpose of the bit vector $b_i$
  assign $seq[i]$, $i = 1, \dots, k$, such that $sum[seq[1]] \leq sum[seq[2]] \leq \cdots \leq sum[seq[k]]$
  repeat
      let $done$=TRUE, $h$= 1;
      for $i = 2$ to $k$
          if $((sum[seq[i]] - sum[seq[i-1]]) < c_{seq[i-1]})$
              $done$=FALSE;
              break;
          if $(!done)$
              let $h++$;
              let $sum[i] = 2 \times sum[i] + b_i^{\mathsf{T}} \cdot A_{m-h}$, $i = 1, \dots, k$;
              assign $seq[i]$, $i = 1, \dots, k$, such that $sum[seq[1]] \leq \cdots \leq sum[seq[k]]$
  until ($done$ or $h > m$);
  if $(\mathcal{C}(k^{m-h}) < \mathcal{C}(seq[1]))$
    let $sum[i] = 2 \times sum[i] + b_i^{\mathsf{T}} \cdot A_{m-j}$, $i = 1, \dots, k$ and $j = h, \dots, m$;
  else
    call OnlineGroupingByBitmaps($\langle g_{seq[1]}, \dots, g_{seq[k]} \rangle, \{b_{seq[1]}, \dots, b_{seq[k]}\}$);
End
```

Fig. 5. Bitmap-based Group By, sorted by aggregated value

The algorithm works as follows. In order to determine the order in the sequence, bitmaps, $A_{m-1}, \dots, A_0$, are read one at a time, until there exists a total ordering in $\langle sum[seq[1]], \cdots, sum[seq[k]] \rangle$. If the cost of using the sequence to calculate the first grouping result is larger than the cost of merging the rest $(m - h + 1)$ bitmaps, $A_{m-h+1}, \dots, A_0$, into $sum[i]$ $(i = 1, \dots, k)$, then the rest

$(m - h + 1)$ bitmaps, $A_{m-h+1}, \ldots, A_0$, are used to produce the final result, else call Algorithm 1 to perform the grouping.

6 Conclusions

Bitmap indexing has rapidly become a standard feature of Data Warehouse platforms. The previously reported use of bitmap indexing has been mostly for search and select processing with particular emphasis on the optimization of individual operations. In this paper we have taken a broader view and have shown how a bitmap-enabled query processor can exploit bitmap indexes to enforce other optimization criteria, such as response time rather than query execution time. To support this view we have developed and presented algorithms that use bitmap indexing to implement the Group-By operator in such a way that results can be produced incrementally in the proper sort order. This makes the results available both for pipelining to other operations, such as Join, and to the end-user working online with the Data Warehouse in an interactive mode.

To take full advantage of bitmap indexing, new bitmap-enabled query processors and optimizers are needed. Future work will concentrate on the development of such an optimizer that integrates and exploits the benefits of bitmap indexing across the whole query.

References

1. C. Y. Chan, Y. E. Ioannidis, *Bitmap Index Design and Evaluation*, Proc. of the ACM SIGMOD Conferenece on Management of Data, Seattle, June 1998. 250, 251, 252
2. C. Y. Chan, Y. E. Ioannidis, *An Efficient Bitmap Encoding Scheme for Selection Queries*, Proc. of the ACM SIGMOD Conference on Management of Data, Philadelphia, June 1999. 250, 251, 252
3. S. Chaudhuri, K. Shim, *Including Group-By in Query Optimization*, Proc. of the 20th International Conference on Very Large Data Bases, Santiago, Chile, Sept. 1994. 250, 254, 255
4. G. Graefe, *Query Evaluation Techniques for Large Databases*, ACM Computing Surveys, 25(2):73-170, June 1993. 256
5. J. M. Hellerstein, P. J. Haas, H. J. Wang *Online Aggregation*, Proc. of the ACM SIGMOD Conference on Management of Data, Tucson, Arizona, May 1997. 250
6. P. O'Neil, D. Quass, *Improved Query Performance with Variant Indexes*, Proc. of the ACM SIGMOD Conference on Management of Data, Tucson, Arizona, May 1997. 250, 251, 252
7. M. C. Wu, A. Buchmann, *Encoded Bitmap Indexing for Data Warehouses*, Proc. of the 14th International Conference on Data Eengineering, Orlando, Feb. 1998. 250, 251, 251
8. M.-C. Wu, *Query Optimization for Selections Using Bitmaps*, Proc. of the ACM SIGMOD Conference on Management of Data, Philadelphia, June, 1999. 250, 251
9. Weipeng P. Yan, Per-Lke Larson, *Performing Group-By before Join*, Proc. of the 10th International Conference on Data Engineering, Houston, Feb. 1994. 250, 255

On Interactive Computation: Intelligent Tutoring Systems
(Extended Abstract)

Max H. Garzon and The Tutoring Research Group[⋆]

The University of Memphis, Memphis TN, 38152
www.psyc.memphis.edu/trg/trg.htm

Abstract. This talk will give an overview of an interdisciplinary research project being developed at The University of Memphis, led by a team of computer scientists, psychologists, and educators. The project's goal is to research and develop prototypes for an *intelligent autonomous software agent* capable of tutoring a human user on a narrow, but fairly open, domain of expertise. The chosen prototype domain is computer literacy. The agent interacts with the user in natural language and other modalities. It receives input in typewritten form, possesses a good deal of syntactic and semantic capabilities to interpret inputs in context relevant fashion, select appropriate responses (short feedback, dialog moves), and completes the dialog cycle in multimodal form (feedback delivered in short spoken expressions and/or facial gestures, spoken information delivery and pointing to appropriate illustrations, animations, etc.). The performance of the agent is expected to be consistent with the level of performance of untrained human tutors. The talk will give a brief overview of the overall architecture of the tutor, explore some of the challenges and tools that have been used in solving them, and provide a demo of the current version, *AutoTutor*, with an emphasis on the multimodal delivery of the dialog cycle.

AutoTutor can be seen as consisting of two interacting large modules: *language* and *sizzle*. The language module is there to understand the student's input (text from keyboard at present). It currently consists of several submodules, including parsers (for syntactic analysis), latent semantic analysis (LSA, for meaning extraction), speech act classifier (has the student given an answer or asked a question?) and dialog moves (how should AutoTutor respond to the student's input?). The sizzle module is there to take the abstract description of

[⋆] This group consists of over twenty faculty and students in psychology, computer science and education funded by the National Science Foundation. They include currently: Pat Chipman, Scotty Craig, Rachel DiPaolo, Stan Franklin, Max Garzon, Barry Gholson, Art Graesser, Doug Hacker, Derek Harter, Xiangen Hu, Bianca Klettke, Roger Kreuz, Kirsten Link, Zhijun Lu, William Marks, Brent Olde, Natalie Person, Victoria Pomeroy, Katja Wiemer-Hastings, Peter Wiemer-Hastings, Holly White, and several new students.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 261–263, 1999.

the pedagogically best reponse in the dialog turn to give the student, as determined by the language modules, and enact it, i.e. give it back to the student in multimodal and ergonomically natural form. This module is enacted by a *talking head* and includes back-channel feedback (visual gestures including emotional reactions for the students), verbal short feedback (e.g., appropriate frozen expressions from a selected repertoire), and/or information splices pointing to appropriately selected pictures and animations illustrating the target concept.

We use two approaches to talking head design embodying sizzle delivery. In the so-called 'canned' approach, the sizzle is put together from picture files shown in appropriate succession. These files need to be choreographed manually (currently using the Microsoft's *agent* program [6]). This approach is potentially computationally taxing because of timewise expensive frequent access of external memory. On the other hand, it allows high quality artistic design, rendering, and rapid prototyping. In the second so-called 'on-line' approach, the sizzle generates entirely on the fly the graphics, sketches, and animations required to deliver each Auto Tutor's dialog turn. This approach would allow AutoTutor to react in real-time when integrated with the language submodules. On the other hand, it presents a challenge to computer scientists to integrate standards (such as MPEG4) for face and body part graphics and animation, with the relevant output from the language module in order to show naturalistic facial expressions and gestures that convey the target feedback (emotions, gestures). In either case, the sizzle is context sensitive, yet can be implemented independently and autonomously by AutoTutor. The driving goal is to make AutoTutor a truly autonomous agent able to visualize graphically the smarts derived from its natural language prowess. Studies are in progress to evaluate the effectiveness of AutoTutor.

There are several questions posed by this type of interactive agents. The computational issues, already mentioned above, concerning rendering graphics and animation involved in facial and bodily features are certainly of research interest. Despite a large literature on animating human-like figures (see Badler et al, 1993; Perlin, 1995 for example) and even faces (the MPEG standard [7], for example), placing face and pointing arms in a tutorial context where the focus of the user's attention is centered elsewhere (comprehension, speech inderstading of a synthetic voice, attentiong shifting back and forth between the agent and illustrations) and the agent must process a number of many other tasks (natural language among others), alters dramatically the space of feasible solutions. The most challenging questions concern the basic principles that govern agent-human interactions. Despite documented evidence that humans tend to think of and treat virtual agents in the same way they do humans (Reeves and Nash, 1996), it is not clear to us that the same kind of devices humans use for effective communication among humans (gestures Krauss, 1998; egocentric generation Keysar, Bar & Horton, 1998, for example) are the best ways for a virtual agent to communicate to and with a human. Vice versa, it seems clear that limitations in the perceptual apparatus of virtual agents (keyboards, or even speech recognition) will force humans and researchers to adapt and develop more effective strategies for interaction with this new kind of intelligent agents. Integrating appropriate computational, cognitivist, and artistic constraints at the right level of granularity in agents of this type is an open topic for interdisciplinary research.

References

1. N. Badler, C. Phillips, B. Webber. *Simulating Humans: Computer Graphics, Animation, and Control*. Oxford University Press, 1993.
2. S. D. Craig, B. Gholson, M. Garzon, X. Hu, W. Marks, P. Wiemer-Hastings, Z. Lu, and The Tutoring research Group. Auto Tutor and Otto Tudor. *Int. Conf. on Artificial Intelligence in Education*, Le Mans, France, July 1999.
3. B. Keysar, D. J. Barr, and W. S. Horton. The Egocentric Basis of Language Use: Insights from a Processing Approach. *Current Directions in Psychological Science* **7**:2 (1998), 46-50.
4. R. M. Krauss. Why do we Gesture When We Speak? *Current Directions in Psychological Science* **7**:2 (1998), 54–60.
5. L. McCauley, B. Gholson, X. Hu, A. Graesser, and the Tutoring research Group. Delivering Smooth Tutorial Dialog Using a Talking Head. *Workshop on Embodied Conversational Characters*, S. Prevost and E. Churchill (Eds)., Tahoe City, CA, 1998.
6. Microsoft Agent 2.1. Microsoft Corporation. http://www.microsoft.com/intdev/agent/. 262
7. The Moving Pictures Experts Group. Document MPEG96/N1365 (draft): Face and Body Definitions and Animation Parameters. http://drogo.cslet.stet.it/mpeg/chicago/animation.htm. 262
8. K. Perlin. Real-time Responsive Animation with Personality. *IEEE Trans. on Visualization and Computer Graphics* **1**:1 (1995).
9. B. Reeves and C. Nash. *The Media Equation. How People Treat Computers, Television and New Media like Real People and Places.* Cambridge University Press, New York, 1996.
10. P. Wiemer-Hasting, A. C. Graesser, D. Harter, and the Tutoring research Group. The Foundations and Architecture of AutoTutor, in preparation.

Coherent Concepts, Robust Learning

Dan Roth and Dmitry Zelenko

Department of Computer Science
University of Illinois at Urbana-Champaign, USA
{danr,zelenko}@cs.uiuc.edu
http://L2R.cs.uiuc.edu/ danr

Abstract. We study learning scenarios in which multiple learners are involved and "nature" imposes some constraints that force the predictions of these learners to behave coherently. This is natural in cognitive learning situations, where multiple learning problems co-exist but their predictions are constrained to produce a valid sentence, image or any other domain representation.

Our theory addresses two fundamental issues in computational learning: (1) The apparent ease at which cognitive systems seem to learn concepts, relative to what is predicted by the theoretical models, and (2) The robustness of learnable concepts to noise in their input. This type of robustness is very important in cognitive systems, where multiple concepts are learned and cascaded to produce more and more complex features.

Existing models of concept learning are extended by requiring the target concept to *cohere* with other concepts from the concept class. The *coherency* is expressed via a (Boolean) constraint that the concepts have to satisfy. We show how coherency can lead to improvements in the complexity of learning and to increased robustness of the learned hypothesis.

1 Introduction

The emphasis of the research in learning theory is on the study of learning *single* concepts from examples. In this framework the learner attempts to learn a single hidden function from a collection of examples (or other, more expressive, modes of interaction) and its performance is measured when classifying future examples. The theoretical research in this direction [19,21] has already proved useful in that it has contributed to our understanding of some of the main characteristics of the learning phenomenon as well as to applied research on classification tasks [5,8].

One puzzling problem from a theoretical and a practical point of view, is the contrast between the hardness of learning problems – even for fairly simple concepts – as predicted by the theoretical models, and the apparent ease at which cognitive systems seem to learn those concepts. Cognitive systems seem to use far less examples and learn more robustly than is predicted by the theoretical models developed so far.

In this paper we begin the study of a new model within which an explanation of this phenomenon may be developed. Key to this study is the observation that

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 264–276, 1999.

cognitive learning problems are usually not studied in isolation. Rather, the input is observed by multiple learners that may learn different functions on the same input. In our model, the mere existence of the other functions along with the constraints Nature imposes on the relations between these functions – all unknown to the learner – contribute to the effective simplification of each of the learning tasks.

Assume for example that given a collection of sentences where each word is tagged with its part-of-speech (pos) as training instances, one wants to learn a function that, given a sentence as input, predicts the pos tag of the ith word in the sentence. E.g., we would like to predict the pos tag of the word `can` in the sentence `This can will rust`[1]. The function that predicts this pos may be a fairly complicated function of other tokens in the sentence; as a result, it may be hard to learn. Notice, however, that the same sentence is supplied as input to the function that predicts the pos of the word `will` and that, clearly, the predictions of these functions are not completely independent. Namely, the presence of the function for `will` may somewhat constrain the function for `can`. For example, the constraint may be that these functions never produce the same output when evaluated on a given sentence. This exemplifies our notion of *coherency*: given that these two functions need to produce coherent outputs, the input sentence may not take *any* possible value in the input space of the functions (that it could have taken when the function's learnability is studied in isolation) but rather may be restricted to a subset of the inputs on which the functions outcomes are coherent. There exists several possible semantics for the coherency conditions and here we present only the one that we find most promising in that we can present results that indicate that the task of learning a concept f becomes easier in these situations.

A fundamental question in the study of learning is that of data preparation. In machine learning it is often found that in order to ensure success at a new learning task considerable effort has to be put into creating the right set of variables, and into eliminating ones if there are large numbers of these. In cognitive learning, on the other hand, there is no evidence for the existence of explicit methods for achieving these ends. The learning process appears to proceed with the set of previously known functions as the set of variables and overcome these problems implicitly.

The ability to chain predictors and perform inferences that are based on learned functions is the second fundamental issue that the coherence assumption contributes to. To study this we define the notion of robustness of learned concepts. In particular, we are concerned with robustness of learnable concepts to attribute noise. This type of robustness is important in cognitive systems, where multiple concepts are learned and "chained" [20,12,16]. Namely, the output of one learned predictor may be used as input to another learned predictor. Thus errors in the output of one predictor translate to attribute noise in the input to another. Therefore, predictors have to tolerate this noise; we show that

[1] This may not be the exact way one chooses to model the problem [18]. However, this is a reasonable abstraction that helps deliver the intuition behind our point of view.

learning coherent concepts results in robust concepts and briefly discuss relations to large margin classification and future work.

The rest of the paper is organized as follows. In Section 2 we describe the standard learning models and delineate the notion of concept *coherency*. Section 3 defines a preliminary semantics of concept coherency and studies its implications. In Section 4 we define the main semantics of concept coherency. We then analyze learning coherent linear separators in Section 4.1. We show that in the new model we can achieve a significant reduction in the mistake bound for the Perceptron. We also investigate the structural properties of coherent linear separators and relate the properties to the mistake bound reduction. In Section 5 we study the relationship between coherency and robustness to attribute noise. First, we introduce a noise model that allows noise to be present when the learned hypotheses are being evaluated and define a robustness condition that guarantees noise tolerance in this model. Finally, we show that coherency entail the robustness condition, thus making l earned concept more robust.

2 Preliminaries

As in the traditional models, the learning scenario is that of concept learning from examples, where a learner is trying to identify a concept $f \in \mathcal{F}$ when presented with examples labeled according to f. We study learning in the standard pac [19] and mistake bound [13] learning models. It is well known that learnability in the pac model depends on the complexity of the hypothesis class. Specifically, it is equivalent to the finiteness of the VC-dimension [22], a combinatorial parameter which measures the richness of the function class (see [21,11] for details). Moreover, it is known [3,6] that the number of examples required for learning is linear in the VC-dimension of the class. Mistake bound learning is studied in an on-line setting[13]; the learner receives an instance, makes a prediction on it, and is then told if the prediction is correct or not. The goal is to minimize the overall number of mistakes made throughout learning process.

The usual way to constrain the learning task is to explicitly restrict the concept class. Here we are mostly concerned with the case in which the restriction is imposed implicitly via interaction between concepts. More precisely, we are interested in a learning scenario in which there exist several concepts $f_1, f_2, \ldots, f_k$ from the concept class $\mathcal{F}$. Let $g\colon \{0,1\}^k \rightarrow \{0,1\}$ be any Boolean function of k variables. The notion of *coherency* we study is formalized by assuming that the concepts $f_1, f_2, \ldots, f_k$ are subjected to a *constraint* g. In all cases, however, we are interested in learning a single function $f_1 \in \mathcal{F}$ under these conditions.

3 Class Coherency

For purposes of illustration we first explore an overly strong notion of coherency, which leads to a restriction on the function class. This is relaxed in the next section and yields the main definition.

Let $\mathcal{F}$ be a concept class over X. The direct k-product $\mathcal{F}^k$ of the concept class $\mathcal{F}$ is the set $\mathcal{F}^k = \{f : f = (f_1, \ldots, f_k), f_i \in \mathcal{F}, i = 1, \ldots, k\}$. Therefore, if $f \in \mathcal{F}^k$, $f \colon X \rightarrow \{0,1\}^k$. Thus, learning k functions with a binary range can be reduced to learning a single function with range $\{0, \ldots, 2^k - 1\}$.

A theorem in [1] states that this transformation (and its inverse) preserves PAC learnability[2].

Theorem 1. *[1]. $\mathcal{F}^k$ is learnable iff $\mathcal{F}$ is learnable.*

Definition 1 (Class Coherency). *Let $\mathcal{F}$ be a concept class and $g \colon \{0,1\}^k \rightarrow \{0,1\}$ a Boolean constraint. $\mathcal{F}_g^k \subseteq \mathcal{F}^k$ is a coherent collection of functions if $\mathcal{F}_g^k = \{(f_1, \ldots, f_k) \in \mathcal{F}^k : \forall x \in X, (g(f_1(x), \ldots, f_k(x)) = 1)\}$.*

Intuitively we can think of g as reducing the range of functions in $\mathcal{F}^k$. That is, if $Y = g^{-1}(1)$, then we do not care about elements $f \in \mathcal{F}^k$ for which $range(f) \not\subseteq Y$.

The observation that a constraint g reduces the range of the functions in $\mathcal{F}^k$ leads to the following sample size bound for pac-learning $\mathcal{F}^k$ which is immediate from the results of [1].

Theorem 2. *Let $m = |g^{-1}(1)|$. Then, the pac learning sample complexity of $\mathcal{F}_g^k$ is $O(\frac{1}{\epsilon}(d(\log m) \log \frac{1}{\epsilon} + \log \frac{1}{\delta}))$ where d is any appropriate capacity measure of $\mathcal{F}_g^k$.*

Example 1. Let $\mathcal{F}$ be the class of axis-parallel rectangles inside $[0,1]^2$. Let $g(f_1, f_2) \equiv (f_1 \neq f_2)$. Then $\mathcal{F}_g^2$ is the class of the pairs (f_1, f_2) of axis-parallel rectangles, where f_1 is the complement of f_2 in $[0,1]^2$. Note that in this case $\mathcal{F}_g^2$ is a class of functions with the binary range $\{01, 10\}$. For binary-valued functions, the appropriate capacity measure of Theorem 2 is the VC-dimension of $\mathcal{F}_g^2$. It is not difficult to see that three points can be shattered by the concept class, but no four points can. Therefore, $VCD(\mathcal{F}_g^2) = 3$; however, $VCD(\mathcal{F}) = 4$ and, hence, Theorem 2 implies that the sample complexity of learning the concept class $\mathcal{F}$ alone is greater than the sample complexity of learning it in the presence of other functions when they are all constrained by g. Thus, adding more concepts may make learning easier.

While definition 1 captures the simultaneous nature of the learning scenario, it is still restrictive in that it imposes global constraints on all the k functions. We would like to relax this further and emphasize that we are interested in learning a single function; say, f_1. We would like to study how the learnability of this function is affected by the presence of the other functions and the requirement that they behave coherently. In the next section we suggest the main definition of this paper.

[2] PAC learnability for multi-valued functions is shown to be characterized by the finiteness of a capacity measure of a function class, see [1] for details.

4 Distributional Coherency

In the previous section we removed from $\mathcal{F}^k$ any f, such that $g(f(x)) = 0$ *for some* $x \in X$. Now, for each $f \in \mathcal{F}^k$, we simply restrict the domain of f to X', where $\forall x \in X', g(f(x)) = 1$. Formally,

Definition 2 (Distributional Coherency). *Given a Boolean constraint g and a class $\mathcal{F}$ of functions, we define the class of g-coherent functions $\mathcal{F}_g^\star$ to be the collection of all functions $f^\star \colon X \to \{0,1\}^k \cup \{\star\}$ defined by*

$$f^\star(x) = \begin{cases} f(x) & \textit{if } g(f(x)) = 1 \\ \star & \textit{otherwise} \end{cases}$$

We interpret the value of "$\star$" as a forbidden value for the function f. In this way we restrict the domain of f to the subset X' of X satisfying the constraint g.

The constraint semantics in Def. 1 is stronger (more restricting) than the one above. To see that let, e.g., $\mathcal{F}$ be the class of (non-identically false) monotone DNF, and g is $(f_1 \neq f_2)$. Then, $\mathcal{F}_g^2$ is empty, because $f_1(1) = 1 = f_2(1)$, for any $f_1, f_2 \in F$. But, in $\mathcal{F}_g^\star$, we simply restrict the domain of each f_1, f_2 to the non-overlapping areas of f_1, f_2.

In the pac learning model the above constraint can be interpreted as restricting the class of distributions when learning a function $f_1 \in \mathcal{F}$. Only distributions giving zero weight to the region $X \setminus X'$ are allowed. We formalize this by introducing the distribution-compatible learning framework.

Definition 3. *Let $\mathcal{F}$ be a class of Boolean functions over X. Let $f_1, \ldots, f_k \in \mathcal{F}$ be subjected to a constraint g. Then, a distribution D over X is said to be f_1-*compatible *w.r.t to $f_2, \ldots, f_k \in \mathcal{F}$ and g, if $D\{x : f^\star(x) = \star\} = 0$. We denote by $\mathcal{D}_{f_1}$ the class of all f_1-compatible distributions (w.r.t to $f_2, \ldots, f_k \in \mathcal{F}$ and g).*

Note that the restriction on the domain of the target function can be arbitrary rather than enforced by a particular Boolean constraint. However, we are mostly interested here in the case in which restrictions naturally arise from constraints on a collection of functions.

To motivate investigation into the gain one might expect to have in this learning scenario, consider the following example.

Example 2. Let $\mathcal{F}$ be the class of disjunctions. Consider learning f_1 from examples, in the presence of f_2 and the constraint $g \equiv (f_1 \neq f_2)$. Suppose that both f_1 and f_2 include a literal l. The constraint implies that X' does not contain examples where l is 1 (otherwise, both f_1 and f_2 will be 1 on the examples). Therefore, the constraint effectively reduces the size of the target disjunction f_1 since the existence of literals common to f_1 and f_2 in the target disjunction is irrelevant to predictions on X'. Thus, if n_1, n_2 is the number of literals in f_1, f_2, respectively, n_c is the number of common literals, then using an attribute efficient algorithm like Winnow to learn f_1 in the presence of f_2 and the constraint g gives a mistake bound of $2(n_1 - n_c)(\log n_1 + 1)$ [13].

This model is a generalization of the Blum and Mitchell[2] model. They study learning two functions f_1, f_2 over different domains (X_1 and X_2, respectively), where the learner sees only pairs $(x_1, x_2) \in X = X_1 \times X_2$ that satisfy $f_1(x_1) = f_2(x_2)$. This is a special case of our model, when $x = (x_1, x_2)$ and the functions f_1, f_2 are defined over subdomains X_1, X_2 rather than the whole X. In example 2, if restricted to monotone disjunctions, we get the domain decomposition for free, because the constraint forces the literal sets of the disjunctions to be disjoint. Thus, by applying the results of [2][3], one can quantify the reduction in the number of examples needed for learning constrained monotone disjunctions.

Next we analyze a more general case of learning in the coherency model.

4.1 Learning Linear Separators

Let $\mathcal{F}$ be the class of half-spaces in R^2 and let g be $(f_1 = f_2)$. f_1 and f_2 are depicted in Figure 1. The arrows point in the direction of the positive half-spaces with respect to the corresponding lines. The constraint g restricts the domains of both f_1 and f_2 to the shaded areas. Therefore, when learning f_1 (and similarly, f_2) we will see examples only from the shaded areas $X' \subseteq X$. For $x \in X'$, $f_1(x) = f_2(x)$. While, in principle, learning f_1 may be hard due to examples nearby the separator, now there are many linear separators consistent with $f_1(x)$. Therefore, at least intuitively, finding a good separator for $f_1(x)$ would be easier. For the case when the linear separator is learned via the Perceptron learning algorithm, we can show the following.

Theorem 3. *Let f_1 and f_2 be two hyperplanes (w.l.o.g, passing through the origin) with unit normals $w_1, w_2 \in R^n$, respectively. Let $\alpha = \cos(w_1, w_2) = w_1 \cdot w_2$. Let $S = S^+ \cup S^-$ be the sequence of positive and negative examples so that $\forall x \in S, f_1(x) = f_2(x)$. Let S be linearly separable by both f_1 and f_2 with margins $2\delta_1$ and $2\delta_2$, respectively. If $\sup_{x \in S} |x| < R$ then the number of mistakes the Perceptron makes on S is bounded by $\beta \frac{R^2}{\delta^2}$, where $\beta = \frac{1+\alpha}{2}, \delta = \frac{\delta_1 + \delta_2}{2}$.*

Proof. For a sequence $S = S^- \cup S^+$ we replace each $x \in S^-$ with $-x$. Then the standard Perceptron learning algorithm becomes:

```
$w := (0, \ldots, 0)$
for all $x \in S$ do
  if $w \cdot x < 0$ then
    $w := w + x$
  end if
end for
```

Since S is linearly separable by f_j with margin δ_j:

$$\forall x \in S, w_j \cdot x > \delta_j > 0, j = 1, 2$$

Let w^i be the value of w after i mistakes. Then, if the ith mistake is made on x:

$$|w^i|^2 = (w^{i-1} + x)^2 = |w^{i-1}|^2 + 2(w^{i-1} \cdot x) + |x|^2 \leq |w^{i-1}|^2 + R^2 \leq iR^2,$$

[3] The results in [2] require in addition certain conditional independence assumptions.

where the last inequality results inductively. Also, using a similar argument,

$$w^i \cdot w_1 = (w^{i-1} + x) \cdot w_1 = (w^{i-1}) \cdot w_1 + w_1 \cdot x \geq (w^{i-1}) \cdot w_1 + \delta_1 \geq i\delta_1 \quad (1)$$

Similarly,

$$w^i \cdot w_2 \geq i\delta_2 \quad (2)$$

Note that (1) and (2) hold simultaneously because f_1, f_2 have the same values on x, and, hence, whenever a mistake is made for f_1, a mistake is also made for f_2. It then follows from (1) and (2) that

$$(w^i \cdot (w_1 + w_2))^2 = ((w^i, w_1) + (w^i, w_2))^2 \geq i^2(\delta_1 + \delta_2)^2. \quad (3)$$

We now bound $(w^i \cdot (w_1 + w_2))^2$ from above. By Cauchy-Schwartz:

$$\begin{aligned}(w^i \cdot (w_1 + w_2))^2 \leq |w^i|^2(|w_1|^2 + |w_2|^2 + 2(w_1 \cdot w_2)) = \\ = 2|w^i|^2(1 + \alpha) \leq 2iR^2(1 + \alpha). \quad (4)\end{aligned}$$

Combining (3) and (4),

$$i^2(\delta_1 + \delta_2)^2 \leq 2iR^2(1 + \alpha)$$

Hence,

$$i \leq \frac{1 + \alpha}{2} \frac{R^2}{(\frac{\delta_1 + \delta_2}{2})^2}$$

Recall that while the general Perceptron mistake bound is $\frac{R^2}{\delta^2}$ [15], the mere presence of f_2 and the constraint g improves the mistake bound by a factor of β. As α approaches -1, the shaded regions become smaller and, hence, β approaches 0.

While Theorem 3 shows the gain in mistake bound when learning w_1 (as a function of w_2 and the constraint) it is possible to quantify this gain in an algorithmic independent way by characterizing the set[4] $E(w_1, w_2)$ of linear separators consistent with the imposed constraint.

Given w_2 and the constraint g, denote by $E(w_1, w_2)$ the set of all linear separators that can be learned without any loss in accuracy when the target concept is w_1. Formally (omitting the dependence on g from the notation), for any two vectors $w_1, w_2 \in R^n$, let $X^{'} = \{x : x \in R^n, sgn(w_1 \cdot x) = sgn(w_2 \cdot x)\}$. That is, $X^{'}$ corresponds to the shaded area in Figure 1. Then:

$$E(w_1, w_2) = \{w : w \in R^n, \forall x \in X', sgn(w \cdot x) = sgn(w_1 \cdot x)\}$$

Theorem 4 uses the well-known Farkas' Lemma [14] from linear programming.

[4] The set $E(w_1, w_2)$ depends on the constraint g. The results in this section can be presented for any symmetric constraint on w_1, w_2, but will be presented, for clarity, only for equality.

Lemma 1. *For any matrix $A_{m\times n}$ and $c \in R^n$, exactly one of these conditions hold.*

(1) $\{x : Ax \le 0, c\cdot x > 0\}$ is non-empty; (2) $\{y : A^T y = c, y \ge 0\}$ is non-empty

Theorem 4. $E(w_1, w_2) = \{w : w = aw_1 + bw_2; a, b \in R; a, b \ge 0\}$.

Proof. Denote $W = \{w : w = aw_1 + bw_2; a, b \in R; a, b \ge 0\}$. Clearly, $W \subseteq E(w_1, w_2)$. In order to prove that $E(w_1, w_2) \subseteq W$, we partition X' in two sets

$$\begin{aligned} X'_+ &= \{x : x \in R^n, w_1 \cdot x \ge 0, w_2 \cdot x \ge 0\} \text{ and} \\ X'_- &= \{x : x \in R^n, w_1 \cdot x \le 0, w_2 \cdot x \le 0\} \end{aligned}$$

Observe that $X'_- = \{-x : x \in X'_+\}$. Fix a $w \in R^n$, so that $w \cdot x \ge 0$ on X'_+. Hence, $w \cdot x \le 0$ on X'_-, and $w \in E(w_1, w_2)$. Now apply Lemma 1 with $A = \binom{w_1}{w_2}$ (A is an $2 \times n$ matrix whose rows are w_1, w_2), and $c = w$. Since $w \cdot x \le 0$ on X'_-, (1) is not satisfied; hence, (2) is satisfied, and $w = aw_1 + bw_2$, where a, b are some positive numbers. Thus, $E(w_1, w_2) \subseteq W$.

If we require the members of $E(w_1, w_2)$ to be unit vectors, then unconstrained learning of f_1 can be viewed geometrically as searching for a point on the unit sphere that is close to the target w_1. In the presence of w_2 and the constraint, we have the following corollary.

Corollary 1. *The intersection of $E(w_1, w_2)$ with the unit sphere is a curve on the unit sphere in R^n connecting w_1 to w_2. The length of the curve is $cos^{-1}(w_1 \cdot w_2)$.*

Thus in the presence of w_2 and the constraint, the learning algorithm seeks a point on the sphere that is close to any of the curve points. As we have shown, algorithmically, for the Perceptron, this translates to reducing the mistake bound proportionally to the length of this curve.

5 Robustness

In this section we show that the coherence assumption made in this paper has the effect of making the learned concepts more robust. We start by defining robustness and proving that concepts learned under this model can indeed be evaluated robustly (generalizing previous models of attribute noise); we then show that learning coherent concepts is robust and discuss the relation to large margin theory.

Definition 4 (Attribute Robustness). *For $x, y \in \{0,1\}^n$ let $H(x, y)$ be the Hamming distance between x and y (that is, the number of bits on which x and y differ). Let*

$$S_k = \{x : \forall y,\ \textit{if } H(x, y) \le k \textit{ then } f(x) = f(y)\}.$$

*We say that the pair (D, f) is (ϵ, k)-*robust*, if $D(S_k) > 1 - \epsilon$.*

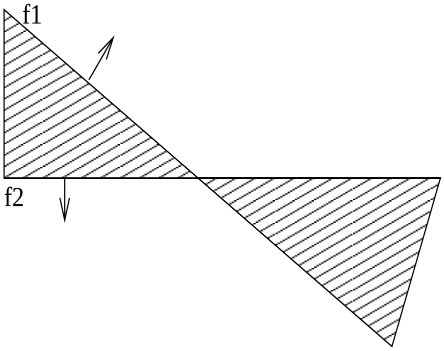


Fig. 1. Constrained Half-spaces in R^2.

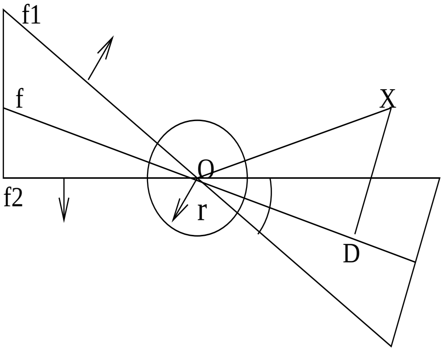


Fig. 2. Constrained Robust Half-spaces

Intuitively, the condition means that w.h.p. all the points in a ball of radius k around any point x have the same label. This can be relaxed by requiring $f(y) = f(x)$ to hold only for a $(1-\gamma)$ portion of the points in the ball $B_k = \{y : H(x,y) \leq k\}$, but we will not discuss this to simplify technical details.

Let f be a concept over $X = \{0,1\}^n$ and let D be a distribution over X. We denote by D^k_{flip} the distribution that results from choosing k bits uniformly and flipping them. It is easy to see that if (D,f) is (ϵ,k)-robust, and $x \in S_k$, then flipping k bits of x does not change the value of f. Hence,

$$error_{D^k_{\mathit{flip}}}(f) \leq D(x \notin S_k) \leq \epsilon$$

and the robustness condition guarantees a small error when evaluating f on the noisy distribution. It follows that if h is an ϵ-good hypothesis for f under D, and if (D,f) is (ϵ,k)-robust, then h is a 2ϵ-good hypothesis under D^k_{flip}.

We note that the distribution D^k_{flip} is an example of an *attribute noise model* [9,4]. These models usually assume the presence of noise in the learning stage and aim at learning a good approximation of the target concept over the original noiseless distribution. However, as can be readily seen (and has been pointed out in [9]), in a more realistic setting in which the learned hypothesis is to be *evaluated* under noisy conditions, this hypothesis may be useless (e.g., consider a simple conjunction or a xor function). The robustness condition defined above guarantees that a hypothesis learned in the presence of noise also performs well when being evaluated under these conditions. This holds for a more general attribute noise model, the product attribute noise, defined as follows. Let D be a distribution on the instance space $X = \{0,1\}^n$. Assume that an attribute i of an example $x \in X$ sampled according to D is flipped independently with probability p_i, $i = 1,\ldots,n$. Denote by $p = \sum_{i=1}^n p_i$ the expected number of bits flipped in an example. We denote by D^p_{flip} the distribution induced this way on X.

Theorem 5. *Let (D, f) be (ϵ, k)-robust. If $k \geq p + \sqrt{2n \ln \frac{1}{\epsilon}}$, then $error_{D^p_{flip}}(f) \leq 2\epsilon$.*

Proof. Let $(x, f(x))$ be an example sampled according to D. Let x' be the result of flipping the bits of x according to the noise scheme described above. Denote by Pr the product distribution induced by the bit flipping. Then we have:

$$\begin{aligned} error_{D^p_{flip}}(f) = D^p_{flip}\{x' : f(x') \neq f(x)\} &= D^p_{flip}\{x' : x \in S_k, f(x') \neq f(x)\} + \\ &+ D^p_{flip}\{x' : x \notin S_k, f(x') \neq f(x)\} \leq \\ &\leq D^p_{flip}\{x' : x \in S_k, f(x') \neq f(x)\} + \epsilon \leq Pr\{H(x, x') > k\} + \epsilon \end{aligned}$$

To bound $Pr\{H(x, x') > k\}$, we let Y be the random variable describing the number of bits flipped in an example. Note that $Y = H(x, x')$ and $E[Y] = p$. Also let $m = \sqrt{2n \ln \frac{1}{\epsilon}}$. Then,

$$Pr\{Y > k\} = Pr\{Y > p + m\} = Pr\{Y - E[Y] > m\} \leq e^{\frac{-m^2}{2n}},$$

where the last inequality follows directly from the Chernoff bounds [10]. Hence, $Pr\{H(x, x') > k\} \leq \epsilon$, and $error_{D^p_{flip}}(f) \leq 2\epsilon$.

Thus, if we have an ϵ-good hypothesis for noiseless distribution, and the target concept with the underlying distribution satisfy the above (ϵ, k)-robustness condition then the hypothesis will also be 3ϵ-good for the noisy distribution.

5.1 Coherency Implies Robustness

We now establish the connection between coherency and robust learning. This is done in the context of learning linear separators learning, as in Section 4.1. As before, the target function is f_1, and we assume the presence of f_2 (w.l.o.g., both f_1, f_2 pass through the origin), and that they are subjected to the equality constraint g. However, here we restrict the domain of f_1 and f_2 to $X = \{0, 1\}^n$. Let D be a distribution over X. We require the distribution to give small weight to points around the origin. Formally, let $B_r = \{x : x \in X, |x| \leq r\}$ be the origin-centered ball of radius r. Then we require D to satisfy $D(B_r) < \epsilon$.

Notice that in general, when learning a single linear separator f, this property of D does not imply that (D, f) is robust. The following theorem shows that with the equality constraint imposed, the property is sufficient to make (D, f) robust.

Theorem 6. *Let f_1 and f_2 be hyperplanes (through the origin) with unit normals $w_1, w_2 \in R^n$, respectively. Let $\alpha = \cos(w_1, w_2) = w_1 \cdot w_2$. Let D be a f_1-compatible distribution (w.r.t f_2 and the equality constraint g) that satisfies $D(B_r) < \epsilon$, where $r > k\sqrt{\frac{2}{1+\alpha}}$. Then, there is a linear separator f, so that $D(x : f(x) \neq f_1(x)) = 0$ and f is (ϵ, k)-robust.*

Proof. (Sketch) The idea of the proof is to exhibit a linear separator f that is consistent with f_1 on D and, for all points lying outside B_r, has a large "margin" separating positive examples from negative ones. Let f be the hyperplane bisecting the angle between f_1 and f_2. That is, $w = \frac{1}{2}(w_1 + w_2)$, where w is the normal vector of f. By theorem 4, $w \in E(w_1, w_2)$; hence, $D(x : f(x) \neq f_1(x)) = 0$. Now fix a point $x \in R^n$, so that $f_1(x) = f_2(x)$ and $x \notin B_r$. Figure 2. is the projection of f_1, f_2, f to the 2-dimensional plane determined by the origin, the point x and x's projection onto f. Then we have that $|XD| = |x| \sin(XOD) \geq r\sqrt{\frac{1+\alpha}{2}}$. If $r > k\sqrt{\frac{2}{1+\alpha}}$, then $|XD| > k$, hence flipping $\leq k$ bits of x will not change the value of f ($|XD|$ is the distance from x to f). Therefore, f is $(0, k)$-robust for any point of the subdomain X' satisfying the constraint and lying outside of the ball B_r. This implies that (D, f) is (ϵ, k)-robust.

We note that the assumption $D(B_r) < \epsilon$ in the Theorem 6 is satisfied if there is a margin r separating positive examples of f_1 from its negative examples [21], so that the weight (with respect D) of examples lying inside the margin is less than ϵ. Also, existence of such a *distributional* margin implies that a sample of examples from the distribution will be linearly separable with margin at least r with high probability, thus guaranteeing that there is a large margin hyperplane consistent with the sample, that has small error with respect to D [7]. In particular, we construct such a hyperplane f in the proof of Theorem 6.

6 Conclusions

This paper starts to develop a theory for learning scenarios where multiple learners co-exist but there are mutual compatibility constraints on their outcomes. We believe that these are important situations in cognitive learning, and therefore this study may help to resolve some of the important questions regarding the easiness and robustness of learning that are not addressed adequately by existing models. In addition, we view this model as a preliminary model within which to study learning in a multi-modal environment. We have shown that within this model the problem of learning a single concept – when it is part of an existing collection of coherent concepts – is easier relative to the general situation. Moreover, this gain is due only to the existence of the coherency, even if the learner is unaware of it.

The results of this paper are restricted mostly to linear separators – not a severe restriction given their universal nature in theory and applications [16,17]. Some of the future directions of this work include the study of more general families of constraints as well as some algorithmic questions that arise from this point of view, including the relations to large margin classification alluded to above.

Acknowledgments

This research is supported by NSF grants IIS-9801638 and SBR-9873450.

References

1. S. Ben-David, N. Cesa-Bianchi, D. Haussler, and P. M. Long. Characterizations of learnability of {0, ... ,n}-valued functions. *Journal of Computer and System Sciences*, pages 74–86, 1995. 267, 267, 267, 267
2. A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proc. of the Annual ACM Workshop on Computational Learning Theory*, pages 92–100, 1998. 269, 269, 269
3. A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–865, 1989. 266
4. S. E. Decatur and R. Gennaro. On learning from noisy and incomplete examples. In *Proc. of the Annual ACM Workshop on Computational Learning Theory*, pages 353–360, 1995. 272
5. H. Druker, R. Schapire, and P. Simard. Improving performance in neural networks using a boosting algorithm. In *Neural Information Processing Systems 5*, pages 42–49. Morgan Kaufmann, 1993. 264
6. A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82(3):247–251, September 1989. 266
7. Y. Freund and R. Schapire. Large margin classification using the Perceptron algorithm. In *Proc. of the Annual ACM Workshop on Computational Learning Theory*, pages 209–217, 1998. 274
8. A. R. Golding and D. Roth. A winnow based approach to context-sensitive spelling correction. *Machine Learning*, 34(1-3):107–130, 1999. Special Issue on Machine Learning and Natural Language. 264
9. S. A. Goldman and R. H. Sloan. Can PAC learning algorithms tolerate random attribute noise? *Algorithmica*, 14(1):70–84, July 1995. 272, 272
10. Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963. 273
11. M. Kearns and U. Vazirani. *Introduction to computational Learning Theory*. MIT Press, 1994. 266
12. R. Khardon, D. Roth, and L. G. Valiant. Relational learning for nlp using linear threshold elements. In *Proc. of the International Joint Conference of Artificial Intelligence*, 1999. 265
13. N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988. 266, 266, 268
14. O. L. Mangarasian. *Nonlinear Programming*. McGraw-Hill, 1969. 270
15. A. Novikoff. On convergence proofs for perceptrons. In *Proceeding of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, 1963. 270
16. D. Roth. Learning to resolve natural language ambiguities: A unified approach. In *Proc. National Conference on Artificial Intelligence*, pages 806–813, 1998. 265, 274
17. D. Roth. Learning in natural language. In *Proc. Int'l Joint Conference on Artificial Intelligence*, pages 898–904, 1999. 274
18. D. Roth and D. Zelenko. Part of speech tagging using a network of linear separators. In *COLING-ACL 98, The 17th International Conference on Computational Linguistics*, pages 1136–1142, 1998. 265

19. L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984. 264, 266
20. L. G. Valiant. Robust logic. In *Proceedings of the Annual ACM Symp. on the Theory of Computing*, 1999. To appear. 265
21. V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995. 264, 266, 274
22. V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its applications*, XVI(2):264–280, 1971. 266

Application of Artificial Neural Networks for Different Engineering Problems

Martin Bogdan and Wolfgang Rosenstiel

Universität Tübingen, Wilhelm-Schickard-Institut für Informatik
Technische Informatik, Sand 13, D-72076 Tübingen, Germany
{bogdan|rosenstiel}@informatik.uni-tuebingen.de
www-ti.informatik.uni-tuebingen.de

Abstract. This paper presents some applications of data and signal processing using artificial neural nets (ANNs) which have been investigated at the University of Tübingen. The applications covering a wide range of different interesting domains: color restoration, gas sensing systems, internet information search and delivery, online quality control and nerve signal processing. The paper presents each application in detail and describes the problems which have been solved.

1 Introduction

The relevancy of artificial neural nets (ANNs) for industrial application rose significantly in recent years, especially in the domain of data and signal processing. ANNs owe this effect to its different abilities as learning using data samples, processing of nonlinear data, parallelism and its insensitivity to noisy data.

This paper comprise a selection of applications of data and signal processing using artificial neural nets which have been recently investigated at the institute. The selected applications covering a wide range of different domains:

- Color restoration of scanned images[1]
- Gas sensing systems[2]
- Internet information search and delivery[3]
- Online quality control of semiconductor chips[4]
- Real time processing of nerve signals[5]

[1] The project *Restoration of Colors in Scanned Images using Artificial Neural Networks* is partly granted by the Daimler-Benz-Stiftung, project #029547.

[2] The project *Molecular Recognition for Analytics and Synthesis using Artificial Neural Nets* is granted by the DFG (Deutsche Forschungsgemeinschaft).

[3] The project *OASIS* (*O*pen *A*rchitecture *S*erver for *I*nformation *S*earch and Delivery) is granted by the European Community under INCO Copernicus project Programme #PL96 1116.

[4] The project *SMART Fabrication-Neural Networks: New production concepts in semiconductor manufacturing* is granted by the Bundesministerium für Bildung und Forschung (BMBF).

[5] The project *INTER* (*I*ntelligent *N*eural In*TER*face) is granted by the European Community under ESPRIT BR project #8897.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 277–294, 1999.

After a brief introduction to artificial neural nets we are concentrating in this paper on the applications of ANNs. In this paper we show the possibility to apply ANNs to different data and signal processing problems whereas the special attributes of ANNs are required.

2 Artificial Neural Nets

Artificial neural nets are information processing systems consisting of several elementary units which are inspired by biological neurons. The elementary unit is called 'neuron' like its biological paragon. Thus, an artificial neuron represents a mathematical model of a biological neuron. Like a biological neural net the ANN transmits the information (activation) between the neurons via unidirectional connections. Due to the topology of the net as well as to the weighting of the connections between the neurons an ANN is capable to represent linear as well as nonlinear function or coherences.

Even if a great number of various types of ANNs are existing, every ANN consists of at least 2 layers: an input layer and an output layer. As shown in Figure 1, most ANNs possess an additional hidden layer. Each layer consists of a number of neurons whereas the number of the input layer is equal to the number of components of the input vector. The number of neurons of the hidden layer as well as the number of the output layer depends on the application.

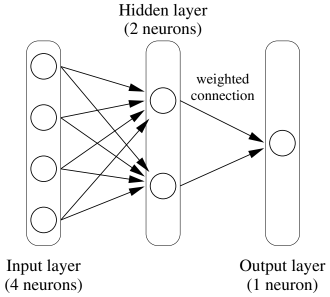


Fig. 1. A simple example of an ANN (feedforward net) consisting of an input layer, a hidden layer and an output layer

Before an ANN can be used for an application, it must be trained. The training is a presentation of data samples to the ANN. Due to special training algorithms the ANN is able to 'learn' the coherences within the presented data samples and changes its weighting in a way to obtain the correct corresponding output.

Depending on the training algorithms the different existing ANNs can be roughly divided into 2 classes: supervised learning and unsupervised learning neural nets. Supervised learning neural nets need an output vector with the desired output to a corresponding input vector. The most known supervised learning nets are probably feedforward nets. Using feedforward nets different training algorithms can be applied. The most common one is the Backpropagation algorithm [1,2] and its variations.

In contrast to the supervised learning nets unsupervised learning nets don't need a desired output vector corresponding to the presented input vector. Representatives of this class are the self-organizing maps [3,4] and ART (Adaptive Resonance Theory) [5].

Which one of the different algorithms is the best to solve a problem depends on the requirements of the application and its data. The choice is a matter of the operator and needs some experience. We have investigated different ANNs for the applications presented in this paper. Main advantages of the ANNs are their abilities as learning of coherences using data samples, processing of nonlinear data, parallelism and its insensitivity to noisy data.

3 Restoration of Colors in Scanned Images

In this chapter we present an approach to color restoration in scanned images, exploiting this capability. The scanner is a device which represents the entire range of color distortions taking place in color image processing. The distortions appear as non-linear spatial defects in the color gamut caused by inaccuracies of the scanner filters and sensors. The problem of restoration can be represented as an approximation of an inverted relationship between the color primaries.

3.1 Color Representation, Distortions and Approach

Color calibration in the area of computer graphics and the printing industry is a crucial issue. Due to inaccuracies and instabilities of the electronic elements and physical properties of dye or ink, the goal to match exactly the initial image remains very difficult to achieve. A challenging task is to compensate these distortions numerically in the stage of image processing.

Figure 2 illustrates the scale of color distortions. The distortions appear as spatial defects negligible around the grey axe and dramatically increasing to the frontiers of the color gamut. An important fact is, that despite this severe deformation of the color gamut, especially strong close to its frontiers, no hysteresis was observed, therefore we can still recover the geometrical color variation.

A wide range of publications deal with the application of artificial neural networks to color processing. Conventional color restoration methods like polynomial regressions, look-up tables, linear interpolations etc. [6] are shown to be often a compromise between fine color resolution and time demands. This is caused by the high non-linearity of distortions associated with a very large number of pixels per image. A comparative study between a neural network with Cascade Correlation learning architecture and polynomial approximations ranging from a 3-term linear fit to a 14-term cubic equation demonstrates that

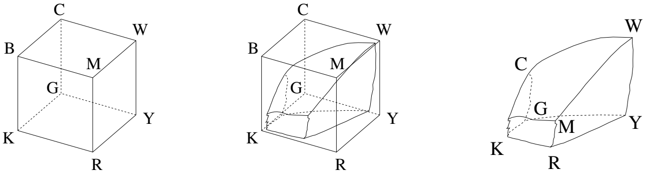


Fig. 2. Edges of the RGB gamut: original colors (left), combined original and scanned (centre) and scanned only (right). Gamut corners are marked according to the pure colors they contain

the neural net outperforms these polynomial approximations [7]. Neural networks have been applied to the prediction of the color ink recipe [8]. In the next section we present our approach to data acquisition, selection of network architecture and learning method and discuss their effectiveness in respect to this particular problem.

3.2 Color Data Acquisition and Neural Network Architecture

Preparation of data, which will be fed into neural network is a very important step. Adequate representation and preprocessing (filtering, dimension reduction, scaling etc.) of input data can be of dramatic influence to the success of neural network application. We have used medium-end laser printer and flatbed scanner for data acquisition. Each primary color was represented by 11 uniform steps. Thus the entire RGB space was represented by altogether 1331 color patches. The RGB values of the scanned color patches formed the input data set for the neural network, and the RGB values of the original colors in turn formed the output set. The data cycle is presented in Figure 3.

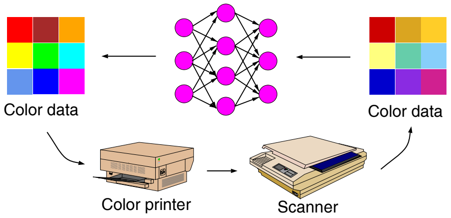


Fig. 3. The data cycle: Color tables are printed, paper samples are scanned – the difference between the resulting data set and initial colors is used for training the ANN

Comparing different learning methods such as backpropagation (sigmoid and Gauss functions), Quickprop, Resilient Propagation (Manhattan-Training) and an interpolating SOM (I-SOM [9]) on a given data set provided us with following considerations: The error gradient descent process of backpropagation was unacceptably slow. Quickprop has shown a quick start in reducing the error but also easily overtrains. The Rprop algorithm demonstrated good convergence as well as good generalization on testing data. In attempts to minimize the error in the testing data we have varied the number of hidden neurons. An increase of the network size beyond 20+20+20 does not result in substantial changes of error.

In the case of Counterpropagation, an interpolation version was used [9]. The Kohonen layer was preset with vectors from the RGB space, corresponding to the scanned color patches and the Grossberg layer with original values for those patches. 216 Codebook vectors were chosen, so that the RGB square was homogeneously represented. In the best result, the RMS error for the test data performed in the same range as for feedforward methods.

3.3 Discussion

We have described an application of neural networks to restoration of colors in scanned images. This approach proves to be feasible and Resilient Propagation method shows a good balance of numerical and qualitative characteristics of the restoration. The neural network offers a promising solution for the problem of color restoration in digital images.

4 Data Evaluation Method for Hybrid Gas Sensing Systems

Chemical and biochemical sensors are used for a broad spectrum of applications. They may be applied in areas like environmental monitoring, process control, medical and quality analysis. Arrays of these sensors are usually called electronic noses [10,11,12]. The sensor signals serve as input to the feature extraction and a subsequent pattern recognition algorithm or multicomponent analysis. In order to increase their performance many efforts were made starting with improvements concerning sampling, filtering, preconditioning via advanced sensor technology up to better feature extraction and data evaluation methods.

As a reference system, the commercial hybrid gas sensing system Moses II (Modular Sensor System II) has been used for qualitative and quantitative analyses. The standard setup uses a Quartz Crystal Microbalance Module (QMB-Module) and a Metal Oxide Module (MOX-Module). Each module is equipped with eight Sensors. For each sample the sensor responses are recorded while the analyte proceeds through the modules. Depending on the flow, about 500 discrete values are recorded for each sample and each sensor. Usually the maximum response of a sensor is used as single feature.

For improvements in the pattern recognition and multivariate data analysis part, common data evaluation methods have been tested systematically for various model systems including coffee and maize oil.

Quite typical for applications of electronic noses, a small number of samples is taken as reference data and a large set of test data has to be classified or predicted. It is well known that this can get many evaluation methods into trouble, especially if high dimensional feature vectors are used.

4.1 Selected Case Study

4.1.1 Qualitative Analysis As a specific example for applications in the food industry several coffee brands available on the German market were analyzed with Moses II. Examined brands included Hanseatica “Espresso dunkel”, Jacobs “Krönung” (pack 1), Jacobs “Krönung” (pack 2), Melitta “Harmonie”, Melitta “Auslese”, Jacobs “Mein Mild’Or” and Tchibo “Feine Milde”.

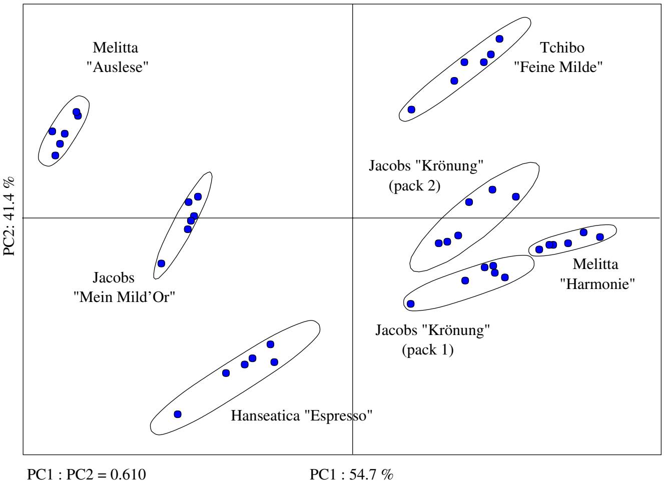


Fig. 4. PCA Scores Plot of five different coffee brands available on the German market

For each brand six samples were measured. 16 sensors were used with the peak height as single feature. Thus, for each sample we had got a feature vector with 16 components. The resulting PCA Scores-Plot for this application is shown in Figure 4.

Different ratios of training data to test data were used to evaluate the recognition performance of each method. The results are presented in Table 1.

Table 1. Recognition performance of selected data evaluation methods

Part of the data that has been used as reference data	Number of correctly classified test samples (in %)				
	KNN	BPN	SOM/G	RBF	ART
2/3	100 %	100 %	97,6 %	97,6 %	90,5 %
1/2	100 %	97,6 %	97,6 %	97,6 %	78,6 %
1/3	87,5 %	97,6 %	100 %	95,2 %	78,6 %

For artificial neural networks size and topology are crucial parameters. The number of input and output units is fixed by the number of sensors (16) and classes (7). Different values for the hidden layers resp. the size of the feature map were tested systematically for these parameters to obtain optimal results. For Backpropagation (BPN) the best result was achieved with 11 hidden neurons. Now, after the topology had been fixed, the test set was used as validation set to find the optimal number of training cycles to get the best results and to avoid overtraining (early stopping method). For the self organizing feature map several sizes have been tested and the best results were achieved with a map-size of 4 3.

4.1.2 Quantitative Analysis To see how different evaluation methods perform in predicting concentrations of gases they were applied to a measurement of gas mixtures containing toluene, octane and propanol. The data set covered mixtures with same concentrations of each gas (100 ppm, 200 ppm or 300 ppm) and mixtures with the concentration of one gas slightly increased (by 20 ppm or 50 ppm). Additionally for each gas concentrations between 100 ppm and 350 ppm were measured in absence of the other two gases. For each mixture three samples were taken. This lead to a total of 144 samples.

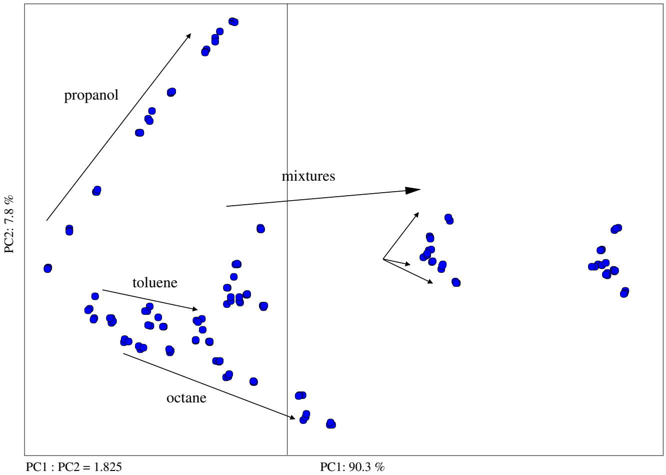


Fig. 5. PCA scores plot of propanol, toluene, octane and their mixtures. Arrows indicate rising concentration (100 ppm to 950 ppm)

The evaluation methods selected for this study were Multivariate Linear Regression (MLR), Principle Components Regression (PCR), Partial Least Squares (PLS), a Counterpropagation Neural Network (SOM/G) and a Standard Backpropagation Neural Network (BPN). The mean prediction error in percent (referring to the total concentration of the mixture) for each method is displayed in Table 2. The number of principle components has been varied between 2 and 16. The number of hidden neurons for the BPN was determined by the same procedure explained for the coffee dataset. The best results were achieved with 22 hidden neurons but even with seven hidden neurons the results of the BPN were much better than the results of the statistical methods like PCR, MLR and PLS.

Table 2. Prediction error of different evaluation methods for gas mixtures in the range of 100 ppm to 950 ppm

Ratio of reference data	mean prediction error				
to test data	MLR	PCR	PLS2	BPN	SOM/G
1/3 : 2/3	5,617 %	4,951 %	4,970 %	2,730 %	0,752 %

4.2 Discussion

BPN and KNN lead to the best results for qualitative analyses. Both allow nonlinear discrimination. KNN is one of the most simple clustering algorithms and therefore easy to implement and easy to use. BPN can also be used for quantitative analyses and can be seen as kind of general purpose neural net. In predicting gas concentrations, it outperformed the statistical methods MLR, PCR and PLS. Despite the fact that small data sets were used, good results were achieved with all covered data evaluation methods.

5 Internet Information Search and Delivery

Another interesting project is the application of ANNs for information search. In frames of the OASIS-project ("Open Architecture for Server for Information Search and Delivery") we develop a multi-server platform that is designed to ideally balance the load of internet search processes. Searching for relevant information in the internet carried out by traditional search engines can not cope with user needs. Delivered results often do not satisfy users as not only relevant, but also lots of irrelevant documents are shown to them.

The OASIS project wants to alleviate internet searching introducing an open architecture server that uses artificial neural networks to cluster HTML documents at the result merging and at the HTML collection description step. User requests will be forwarded to those servers of the server system whose collection topic is most likely to match the user request. Requests from different topic collections will be merged at the server that propagated the user request. This result merging is carried out by a neural network that has exclusively been designed for OASIS project, called *hierarchical radius-based competitive learning (HRCL).*

5.1 Neural Network Clusterisation

5.1.1 Hierarchical Radius-Based Competitive Learning We use a *hierarchical radius-based competitive learning (HRCL)* neural network that has exclusively been developed for OASIS project to accomplish the clustering. It is primarily based on the neural gas approach [13] and uses output neurons without fixed grid dimensionality: Neurons are rearranged due to their distances to the current input every time a new input sample is generated. Second, HRCL uses fixed radii around each neuron and repels those second winners from the current input sample whose radii overlap with the winner's radius like in the rival penalized competitive learning algorithm [14].

Third, HRCL builds a hierarchy of clusters and subclusters in either top-down or bottom-up manner: the first generated top-down hierarchical level consists of detected clusters or cluster prototypes. Every neuron has been learned to represent one prototype. The second level then refines first level clustering using the user-supplied fixed radius and tries to detect subclusters at every first level cluster, and so forth. Initial neuron settings at each hierarchical level are generated due to probability densities at fixed cells in vector space using a cell-like clustering similar to the BANG-clustering system [15].

5.1.2 Advantages of HRCL Conventional statistical clustering methods like single- or one-pass clustering as well as similar heuristic methods are highly dependent on the order of input vectors fed into the system. Conventional neural approaches, above all error minimizing competitive learning algorithms, generally are able to detect major clusters.

Competitive learning methods with a-priori given and fixed output neuron dimensionality like Kohonens Self-Organizing Map (SOM) [3,4] also place neurons to locations with lower probability densities.

Competitive learning without a given network dimensionality like Growing Neural Gas (GNG) [16] use adapted network dimensionality at vector hyperspaces with different 'local' fractal dimension and thus try to circumvent the drawbacks mentioned above. Their granularity of detected clusters is highly dependent on the number of training steps: Depending on the duration of the training, GNG will find either clusters with appropriate cluster centroids or only subclusters, but not both.

Figure 6 shows the training results of HRCL hierarchical clustering on 2-dimensional artificially arranged multi-modal input data using top-down hierarchical refinement. The left picture depicts 3 neurons of the first hierarchical level placed at cluster centroids after 565 HRCL learning steps using a user supplied neuron radius of 0.3. The right picture depicts 5 HRCL neurons of the second hierarchy placed at subcluster centers of the cluster that is defined by one of the first level neurons plus its radius using additional 95 learning steps. HRCL is able to automatically build a hierarchy of clusters, subclusters and so on depending on neuron settings at each level and user supplied radius. For abovementioned input data that consist of 1,800 vectors, HRCL automatically detects 3 hierar-

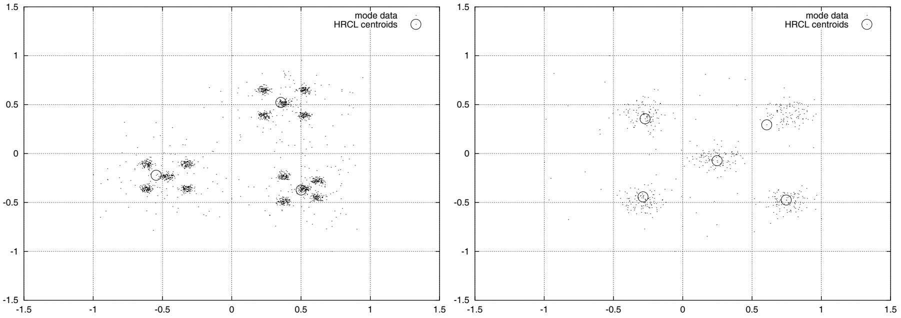


Fig. 6. Left: first level HRCL; Right: second level HRCL

chy levels reflecting the globularity of every top-level cluster and consumes appr. 9 min on a Sun Sparc-Ultra 2.

5.2 Discussion

HRCL is not only able to detect overall clusters, but also produces a hierarchical tree of clusters with appropriate cluster centroids and subclusters, subcluster centroids, and so on – if feasible. Clusters are locations of high probability densities in vector space. That requires coding and compression techniques which are able to generate vectors that lie adjacent in vector space if and only if they share the same topic, clusters will represent HTML documents of the same or similar thematic contents. It is envisaged to use HRCL in order to cluster HTML document collections to obtain an hierarchical tree of clusters and subclusters whereas each cluster will optimally be related to an underlying topical content. User requests will be compared to cluster centroids in order to speed up searching. Even a Yahoo-like browsing facility can be thought of, offering the user to browse from cluster to subclusters and so on to refine her request.

6 Prediction of Functional Yield of Chips in Semiconductor Industry Applications

In semiconductor industry the number of circuits per chip is still drastically increasing, while the dimensions of the chips are continuously reduced. The number of circuits per chip is being doubled approximately every three years. Not only the production cost per chip, but also the number of semiconductor factories increases permanently. All this leads to a strong demand for high quality production on the condition of very low prices. So the importance of quality control and quality assurance is much raised.

One of the most expensive aspects in the manufacturing process of chips are the tests of their functionality. Two different kinds of tests are used during and after the production, one is the process control monitoring (PCM-data) early during the production process and the other test is the ‘functional test’ for testing circuits on finished chips (functional data).

Especially the second test is not only very time consuming and therefore expensive, but also very late, so machine problems are recognized far too late and many defective chips are processed in the meanwhile. For this reason it would be a great improvement if this test could be made obsolete and machine problems or errors could be detected by using PCM-data.

6.1 Data Samples

The aim of this work was the prediction of the functionality of chips by means of PCM-data (yield model). These data samples contain production conditions and physical values like currents or layer thicknesses and are used for the prediction tool. 4860 training vectors of PCM-data have been measured by industrial partners. Each training vector contains 96 components. Thus, the data is not easy to handle for data processing tools.

6.2 Methods

There are many possibilities to reduce the dimension of the training data, but only a few can meet the assumption of a good prediction. In our work we used a regression method and principle component analysis (PCA). After this reduction of parameters we compared the results of different neural network algorithms to achieve best results. In detail, we tried Radial Basis Functions (RBF) [17], Counterpropagation [18] and Feedforward networks like Backpropagation (BackProp) [2] and Resilient Propagation (RProp) [19].

For all networks, we first trained with half of the data samples (2430) and all components (96) and used the rest of the 4860 data samples for the validation set. After that we reduced the components of the input vectors to the number of 10. The main reason was to achieve a good and understandable model for the yield of the chips by means of PCM-data. Last but not least we checked the fitness of our model by forecasting the yield of chips with the reduced data samples. For the output of the networks we had a value between 0 and 100, namely the percentage of non defective chips on each wafer.

6.3 Results

As expected from earlier research results [20,21], the Counterpropagation network (8x12 SOM) with interpolation method proved to be stable and good in both, the prediction of yield with all parameters and with only 10 significant parameters. Surprisingly with RBF the results have been even better and the yield of a wafer has been predicted with an absolute error of about 8 % as shown in Figure 7. Although the algorithm is known to be difficult in forecasting applications with extrapolation requirements, the approximation of the yield was very good for all of the three data sets and not only for the trained data samples, but also for the test data. The reason for this good results seems to be the high number of training samples and the relatively high noise, which has been applied in the training phase.

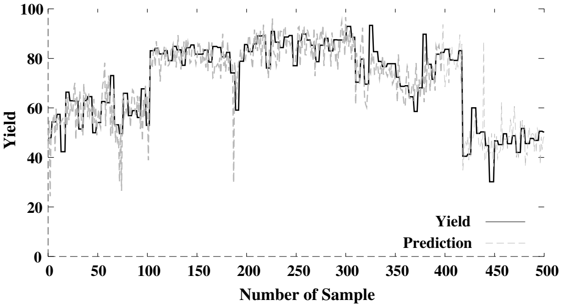


Fig. 7. The result of the prediction with the RBF network and testing with a validation data set

With BackProp (96-15-15-1) and (10-4-1), convergence of the training often was a problem and therefore we were not able to obtain a proper result. The tests with RProp (96-15-1) and (10-3-1) have been converged much faster and the resulting values have been much better, too; the results are in the range of the results of the Counterpropagation algorithm.

If we compare the results of the training with the reduced data set to the training results with all of the 96 parameters we can see a small difference of 1-2 % in the resulting yield. The reduction with PCA seems to deliver the better results for the prediction, but it's much more difficult to find out the original parameters, which are important for the changes in the yield. You only have the eigenvectors of the covariance matrix. So we're going to use the regression method for the reduction of parameters in the future. An overview of the results is given in Table 3.

Table 3. Results of the prediction with different networks and different data samples. The absolute error of the prediction is given in percent

	Counterprop		RBF		BackProp		RProp	
Samples	Training	Test	Training	Test	Training	Test	Training	Test
96 parameters	5.9	9.1	5.7	8.2	13.7	13.9	6.4	9.3
Regression	6.7	9.9	6.2	8.8	10.1	11.3	7.2	10.8
PCA	6.4	9.1	6.4	8.6	9.9	11.1	6.9	10.2

6.4 Discussion

In our work we were able to predict the functional yield of wafers with an absolute error of less than 8 %. This result is precise enough to detect automatically, whether many chips on a wafer are defective or not. We extracted some parameters to have a small and clear yield model for the experts in the semiconductor factory. Checked by some experts the extracted parameters proved to be very important and most obviously responsible for yield variations in the given data samples. Within this work it was possible to forecast the yield of chips very early in the production process; additionally the result provides a tool for fast error recovering by the experts due to a small given number of responsible parameters.

7 Real Time Processing of Nerve Signals for Controlling a Limb Prostheses

ANNs can be used for the signal processing in the medical field as well. Signal processing within the medical field requires a high flexibility and good generalization skills what can be obtained using ANNs. One application of ANNs in the medical field is the INTER-project.

The aim of the INTER-project (*I*ntelligent *N*eural In*TER*face) is to investigate fundamental issues related to the design and fabrication of a new generation of microsystems applicable as neural prostheses. A global overview for a PNS-remoted limb prostheses is given in [22] and is shown in Figure 8.

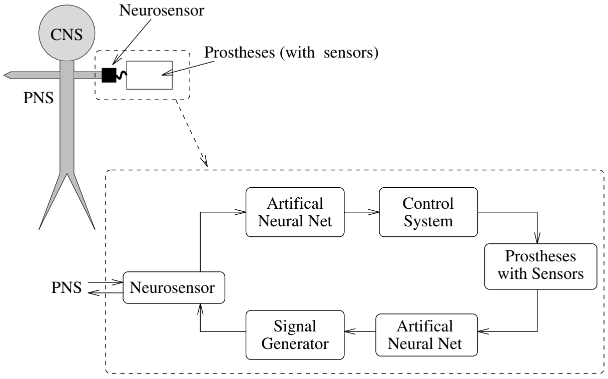


Fig. 8. Scheme Configuration of a bio-neural controlled prostheses

Nerve signals will be recorded and amplified by a regeneration-type neurosensor. Then, an artificial neural net (ANN) is applied which classifies the resulting signals in order to assign certain limb movements to the signal classes. A control unit uses the resulting information to regulate the movement of the prostheses [23,24].

Ideally, the prostheses is equipped with sensors. Signals from the sensors will be processed by an ANN and transmitted via a signal generator and the neurosensor to the peripheral nervous system (PNS) resulting in a kind of natural limb control.

7.1 The Neurosensor

The principle of the implementation and the neurosensor which is used in the INTER-project is shown in Figure 9. Peripheral nerves of vertebrates will regenerate if severed. For this reason, the peripheral nerve can be surgically severed in order to insert the proximal and the distal stump into a guidance channel which envelops the neurosensor.

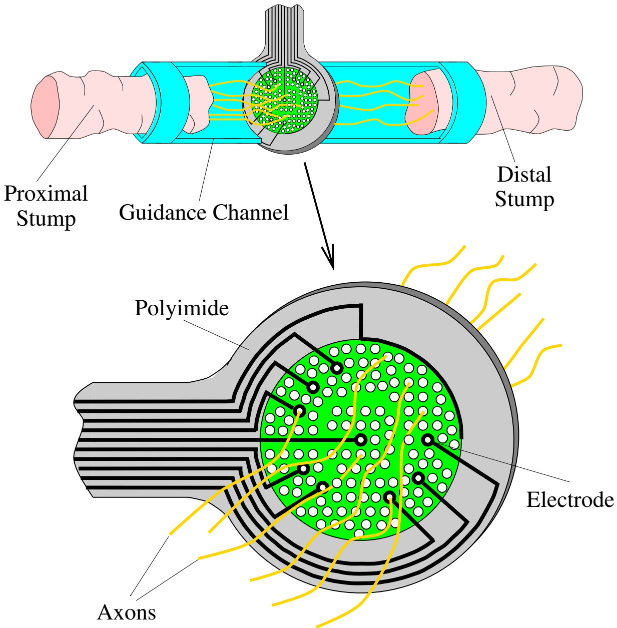


Fig. 9. Implementation scheme for the regeneration-type neurosensor

The sensor is fabricated of polyimide perforated by multiple 'via holes' [25]. The axons regenerate through the via holes from the proximal stump towards the distal stump of the nerve. Nerve signals can be recorded by electrodes, which are enclosing some of the via holes. A circuitry amplifies and preprocesses the nerve signals. The amplified signals are transferred to the units controlling the prostheses as shown in Figure 8.

7.2 Data Set

The data set, which is used for the classification, has been recorded by the Institut für Biomedizinische Technik (IBMT). IBMT has chosen the stomatogastric nervous system (STNS) of the crab *Cancer pagurus* as described in [26].

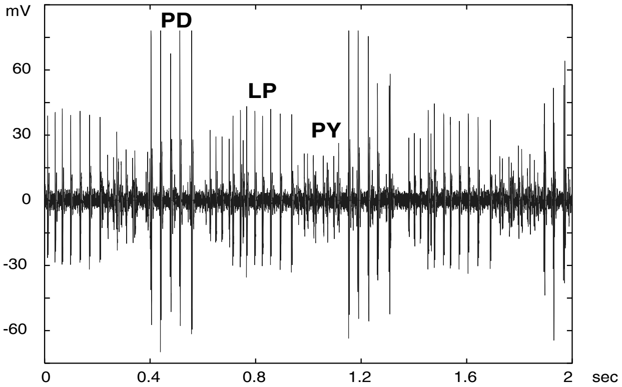


Fig. 10. One detail out of the recordings from the gastric nerve of a crab. PD, LP and PY cells can be easily identified

A typical recorded sequence of the signals are shown in Figure 10. The STNS contains about 30 nerve cell bodies, 24 of which are motorneurons and 6 of which are interneurons. The action potentials corresponding to the PD, LP and PY motorneurons can be easily identified. The durations of the recordings are 24 respectively 40 seconds. The data set were recorded using a sample frequency of 5 kHz. Since this nervous system is very well known, we are able to verify the results obtained by the classification of the SOM for their correctness.

7.3 Classification Using Kohonens SOM

For the classification of the data set a two dimensional SOM [3,4] with 10 neurons in both dimension have been applied. The training data set consists of 2667 vectors with six components describing the shape of a spike of a neuron.

After obtaining a well ordered map, we have identified the clusters within this map using CLUSOT (Cluster in self-organized maps) [27]. Each of the obtained clusters represent one specific signal from an axon respectively from a group of axons (e.g. PD or PY cells). Due to the identification of the clusters within the trained SOM, we can assign an action to each cluster conditioned by a signal of an axon. The obtained clusters are presented in Figure 11.

As mentioned above, each cluster represents the signals from an axon respectively from a group of axons. Every time a nerve signals occurs, it will be classified to its corresponding cluster. Thus we are able to recognize the signals from

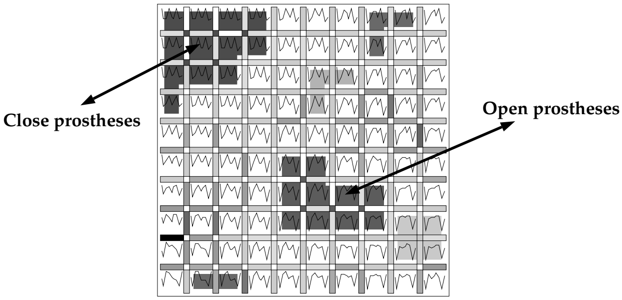


Fig. 11. The trained SOM with obtained clusters. Two of the clusters has been chosen to assign an action of the prostheses

certain axons in order to control the movement of the limb prostheses. In our case we control a commercial artificial hand assigned to the clusters as shown in Figure 11.

7.4 Control Unit of the Prostheses

After classifying the nerve signals to their corresponding clusters the occurance of a nerve signal of a certain cluster must be assigned to its corresponding action. Since the information of the nerve signals are pulse-frequency encoded we have to change into the time domain. The problem within this case is that the occurance of one single nerve signal does not carry useful information. This might be a spontaneous or hazardous signal. For these reasons we have decided to build an integration based signal interpreter to remote the control unit of the prostheses. A detailed description of the control unit is given in [24].

7.5 Discussion

In this chapter, we have presented a signal processing unit basing on artificial neural networks which is able to interprete real nerve signals and to control a limb prostheses in real time. The signal processing system consists of a self-organizing map (SOM) and an integration based signal interpreter. The SOM classifies the nerve signal corresponding to its origin. The integration based signal interpreter controls the movement of the limb prostheses based on the frequency of occurance of the nerve signals. The two parts of the processing unit realize a pulse-frequency decoding of nerve signals.

To summarize, we have presented a real time processing system which is able to remote a limb prostheses due to incoming nerve signals. We have shown that it is possible to interprete nerve signals from recordings of a sum of axons in real time.

8 Conclusion

After a brief introduction to artificial neural nets, we have presented different applications of data and signal processing covering a wide range of different domains from industrial large-scale production up to health care. The presented applications were color restoration, gas sensing systems, internet information search and delivery, online quality control and nerve signal processing. All signal processing problems within these applications could be solved by the use of ANNs whereas we have shown, that different types of ANNs have to be chosen for different requirements.

To conclude, we have presented solutions for various data and signal processing problems. We have shown, that artificial neural nets can be applied not only to academic problems but also to industrial applications. This is due to their abilities like learning of coherences using data samples, parallelism and its insensitivity to noisy data. Especially in the case of nonlinear coherences within the data samples ANNs are extremely advantageous.

Acknowledgment

The authors like to thank Alexei Babanine, Thomas Hermle, Udo Heuser and Lothar Ludwig from Wilhelm-Schickard-Institut für Informatik, Technische Informatik, Universität Tübingen for their contributions to this paper.

References

1. D. E. Rumelhart, G. E. Hinton and R. J. Williams: *Learning representations by back-propagation errors*, Nature 323, pp. 533–536, 1986. 279
2. D. E. Rumelhard and J. L. McClelland: *Parallel Distributed Processing: Explorations in the Microstructure of Cognitron, I & II* MIT Press, Cambridge MA, 1986. 279, 287
3. T. Kohonen: *Self-organized Formation of Topology Correct Feature Maps*, Biological Cybernetics, 43:59-69, 1982. 279, 285, 291
4. T. Kohonen: *Self-Organization and Associative Memory*, Springer Series in Information Sciences, Springer-Verlag, 1989. 279, 285, 291
5. S. Grossberg: *Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors*, Biological Cybernetics 13, pp. 121–134, 1976. 279
6. P.-C. Hung: *Colorimetric calibration for scanners and media*, Proc. SPIE 1448, pp. 164–174, 1991. 279
7. H. R. Kang and P. G. Anderson: *Neural network applications to the color scanner and printer calibrations*, Journal of Electronic Imaging, Vol. 1, No. 2, pp. 125–135, 1992. 280
8. J. M. Bishop, M. J. Bushnell, and S. Westland: *Application of neural networks to computer recipe prediction*, Color Res. Appl. 16, 3–9, 1991. 280
9. J. Göppert: *Die topologisch interpolierende selbstorganisierende Karte in der Funktionsapproximation*, (Ph.D. Thesis), Shaker Verlag Aachen, 1997. 280, 281

10. U. Weimar: *Elektronische Nasen: Gestern, heute, morgen*, Sensoren und Meßtechnik, Number 148 in ITG Fachbericht, pp. 207–227, VDE-Verlag, Berlin, 1998. 281
11. U. Weimar and W. Göpel: *Chemical imaging: Trends in multiparameter sensor systems*, Conf. Proc. EUROSENSORS XI, Warszawa, 1997. 281
12. J. L. Mitrovics: *Entwicklung eines portablen, modularen Sensorsystems zur quantitativen Gasanalyse*, Diploma Thesis, Universität Tübingen, 1994. 281
13. T. M. Martinetz and K. J. Schulten: *A "neural-gas" network learns topologies*, in T. Kohonen, K. Mäkisara, O. Simula, J. Kangas (Eds.): Artificial Neural Networks, North-Holland, Amsterdam, pp. 397–402, 1991. 285
14. L. Xu, A. Krzyzak and E. Oja: *Rival Penalized Competitive Learning for Clustering Analysis, RBF Net, and Curve Detection*, IEEE Transactions on Neural Networks, Vol. 4, No. 4, July 1993. 285
15. E. Schikuta and M. Erhart: *The BANG-Clustering System: Grid-Based Data Analysis*, in X. Liu, P. Cohen and M. Berthold (Eds.): Advances in Intelligent Data Analysis (IDA-97), LNCS 1280, pp. 513–524, 1997. 285
16. B. Fritzke: *A growing neural gas network learns topologies*, in G. Tesauro, D. S. Touretzky and T. K. Leen (Eds.): Advances in Neural Information Processing Systems 7, pp. 625–632, MIT Press, Cambridge MA, 1995. 285
17. T. Poggio and F. Girosi: *A theory of networks for approximation and learning,* A.I. Memo (1140), 1989. 287
18. R. Hecht-Nielsen: *Applications of counterpropagation networks*, Neural Networks, 1(2):131-139, 1988. 287
19. H. Braun and M. Riedmiller: *Rprop: A fast and robust backpropagation learning strategy,* ACNN, pp. 598–591, 1993. 287
20. L. Ludwig, W. Kessler, J. Göppert and W. Rosenstiel: *SOM with topological interpolation for the prediction of interference spectra*, Proc. of EANN '95, Helsinki, Finnland, pp. 379–389, 1995. 287
21. L. Ludwig, U. Epperlein, H.-H. Kuge, P. Federl, B. Koppenhoefer and W. Rosenstiel: *Classifikation of 'fingerprints' of process control monitoring-data with self-organizing maps*, In Proc. of EANN '97, Stockholm, Sweden, pp. 107–111, 1997. 287
22. P. Dario and M. Cocco: *Technologies and Applications of Microfabricated Implantable Neural Prostheses*, IARP Workshop on Micromachine & Systems 1993, Tokyo, 1993. 289
23. M. Bogdan and W. Rosenstiel: *Artificial Neural Nets for Peripheral Nervous System – remoted Limb Prostheses*, Neural Networks & their Applications, pp. 193–202, Nanterre, 1994. 289
24. M. Bogdan and W. Rosenstiel: *Real Time Processing of Nerve Signals for Controlling a Limb Prostheses*, Information Processing in Cells and Tissues, Sheffield, pp. 26–38, 1997. 289, 292
25. T. Stieglitz, H. Beutel and J.-U. Meyer: *A flexible, light-weighted, multichannel sieve electrode with integrated cables for inter-facing regenerating peripheral nerves*, Proceedings of 10th Eurosensors Conference, 1996. 290
26. H. G. Heinzel, J. M. Weimann and E. Marder: *The Behavioral Repertoire of the Gastric Mill in the Crab, Cancer pagurus: An in situ Endoscopic and Electrophysiological Examination*, The Journal of Neuroscience, pp. 1793–1803, April 1993. 291
27. M. Bogdan: *Signalverarbeitung biologischer neuronaler Netze zur Steuerung einer Prothese mit Hilfe künstlicher neuronaler Netze*, Ph.D. Thesis, Universität Tübingen, Cuvillier Verlag Göttingen, ISBN 3-89712-336-3, 1998. 291

Factor Oracle:
A New Structure for Pattern Matching

Cyril Allauzen, Maxime Crochemore*, and Mathieu Raffinot

Institut Gaspard-Monge, Université de Marne-la-Vallée,
77454 Marne-la-Vallée Cedex 2, France
{allauzen,mac,raffinot}@monge.univ-mlv.fr
www-igm.univ-mlv.fr/LabInfo/

Abstract. We introduce a new automaton on a word p, sequence of letters taken in an alphabet Σ, that we call *factor oracle*. This automaton is acyclic, recognizes at least the factors of p, has $m+1$ states and a linear number of transitions. We give an on-line construction to build it. We use this new structure in string matching algorithms that we conjecture optimal according to the experimental results. These algorithms are as efficient as the ones that already exist using less memory and being more easy to implement.

Keywords: indexing, finite automaton, pattern matching, algorithm design.

1 Introduction

A *word* p is a finite sequence $p = p_1p_2 \dots p_m$ of letters taken in an alphabet Σ. We keep the notation p along this paper to denote the word on which we are working.

Efficient pattern matching on fixed texts are based on indexes built on top of the text. Many indexing techniques exist for this purpose. The simplest methods use precomputed tables of q-grams while more achieved methods use more elaborated data structures. These classical structures are: suffix arrays, suffix trees, suffix automata or DAWGs[1], and factor automata (see [11]). When regarded as automata, they accept the set of factors (substrings) of the text. All these structures lead to very time-efficient pattern matching algorithms but require a fairly large amount of memory space. It is considered, for example, that the implementation of suffix arrays can be achieved using five bytes per text character and that other structures need about twelve bytes per text character.

Several strategies have been developed to reduce the memory space required to implement structures for indexes.

[*] Work by this author is supported in part by Programme “Génomes” of C.N.R.S.

[1] DAWGs, Directed Acyclic Word Graphs, are just suffix automata in which all states are terminal states

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM’99, LNCS 1725, pp. 295–310, 1999.

One of the oldest method is to merge the compression techniques applied both by the suffix tree and the suffix automaton. It leads to the notion of compact suffix automaton (or compact DAWG) [5]. The direct construction of this structure is given in [12,13].

A second method to reduce the size of indexes has been considered in the text compression method in [10]. It consists in representing the complement language of the factors (substrings) of the text. More precisely, only minimal factors not occurring in the text need to be considered [9,8]. Which allow to store them in a tree and to save space.

We present in this paper a third method. We want to build an automaton (a) that is acyclic (b) that recognizes at least the factors of p (c) that has the fewer states as possible and (d) that has a linear number of transitions. We already notice that such an automaton has necessarily at least $m+1$ states.

The suffix or factor automaton [4,7] satifies (a)-(b)-(d) but not (c) whereas the sub-sequence automaton [3] satisfies (a)-(b)-(c) but not (d), which makes the problem non trivial.

We propose an intermediate structure that we call the *factor oracle*: an automaton with $m+1$ states that satisfies these four requirements.

We use this new structure to design new string matching algorithms. These algorithms have a very good average behaviour that we conjecture as optimal. The main advantages of these new algorithms are (1) that they are easy to implement for an optimal behaviour and (2) the memory saving that the factor oracle allows with respect to the suffix automaton. The structure has been extended in [2] to implement the index of a finite set of texts.

The paper is structured as follows: Section 2 discusses the construction of the factor oracle, Section 3 describes a string matching based on the factor oracle and shows experimental results, and finally we conclude in Section 4. Proofs of the results presented in the paper may be found in [1]. We now define notions and definitions that we need along this paper.

A word $x \in \Sigma^*$ is a *factor* of p if and only if p can be written $p = uxv$ with $u, v \in \Sigma^*$. We denote $\mathit{Fact}(p)$ the set of all the factors of word p. A factor x of p is a *prefix* (resp. a *suffix*) of p if $p = xu$ (resp. $p = ux$) with $u \in \Sigma^*$. The set of all the prefixes of p is denoted by $\mathit{Pref}(p)$ and the one of all the suffixes $\mathit{Suff}(p)$. We say that x is a *proper factor* (resp. *proper prefix*, *proper suffix*) of p if x is a factor (resp. prefix, suffix) of p distinct from p and from the empty word ϵ.

We denote $\mathit{pref}_p(i)$ the prefix of length i of p for $0 \leq i \leq |p|$.

We denote for $u \in \mathit{Fact}(p)$, $\mathit{poccur}(u,p) = \min\{|z| : z = wuetp = wuv\}$, the ending position of the first occurrence of u in p.

Finally, we define for $u \in \mathit{Fact}(p)$ the set $\mathit{endpos}_p(u) = \{i \mid p = wup_{i+1} \ldots p_m\}$. If two factors u and v of p are such that $\mathit{endpos}_p(u) = \mathit{endpos}_p(v)$, we denote $u \sim_p v$. It is very easy to verify that $\sim_p$ is an equivalence relation; it is in fact the syntaxic equivalence of the language $\mathit{Suff}(p)$.

2 Factor Oracle

2.1 Construction Algorithm

```
**Build_Oracle**($p = p_1p_2 \dots p_m$)
    For $i$ from 0 to $m$
        Create a new state $i$
    For $i$ from 0 to $m - 1$
        Build a new transition from $i$ to $i + 1$ by $p_{i+1}$
    For $i$ from 0 to $m - 1$
        Let $u$ be a minimal length word in state $i$
        For all $\sigma \in \Sigma, \sigma \neq p_{i+1}$
        If $u\sigma \in Fact(p_{i-|u|+1} \dots p_m)$
        Build a new transition from $i$ to $i + poccur(u\sigma, p_{i-|u|+1} \dots p_m)$ by $\sigma$
```

Fig. 1. High-level construction algorithm of the Oracle

Definition 1 *The* factor oracle *of a word* $p = p_1p_2 \dots p_m$ *is the automaton build by the algorithm* Build_Oracle *(Figure 1) on the word* p*, where all the states are terminal. It is denoted by* $Oracle(p)$*.*

The factor oracle of the word $p = abbbaab$ is given as an example Figure 2. On this example, it can be noticed that the word *aba* is recognized whereas it is not a factor of p.

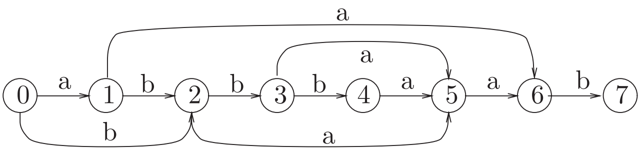


Fig. 2. Factor oracle of *abbbaab*. The word *aba* is recognizes whereas it is not a factor

Note: all the transitions that reach state i of $Oracle(p)$ are labeled by p_i.

Lemma 1 *Let* $u \in \Sigma^*$ *be a minimal length word among the words recognized in state* i *of* $Oracle(p)$*. Then,* $u \in Fact(p)$ *and* $i = poccur(u, p)$*.*

Corollary 1 *Let $u \in \Sigma^*$ be a minimal length word among the words recognized in state i of Oracle(p), u is unique.*

We denote $\min(i)$ the minimal length word of state i.

Corollary 2 *Let i and j be two states of Oracle(p) such as $j < i$. Let $u = \min(i)$ and $v = \min(j)$, u can not be a suffix of v.*

Lemma 2 *Let i be a state of Oracle(p) and $u = \min(i)$. u is a suffix of any word $c \in \Sigma^*$ which is the label of a path leading from state 0 to state i.*

Lemma 3 *Let $w \in Fact(p)$. w is recognized by Oracle(p) in a state $j \leq poccur(w,p)$.*

Note: In lemma 3, j is really less or equal than $poccur(w,p)$, and not always equal. The example given in the Figure 3 represents the automaton *Oracle*$(abbcabc)$, and the state reached after the reading of the word *abc* is strictly less than $poccur(abc, abbcabc)$.

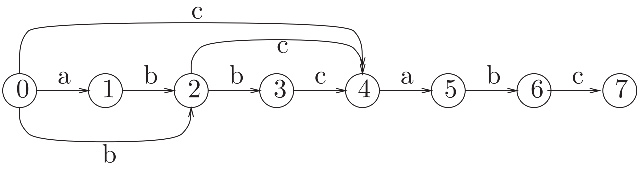


Fig. 3. Example of a factor (*abc*) that is not recognized at the end of his first occurrence but before

Corollary 3 *Let $w \in Fact(p)$. Every word $v \in Suff(w)$ is recognized by Oracle(p) in a state $j \leq poccur(w)$.*

Lemma 4 *Let i be a state of Oracle(p) and $u = \min(i)$. Any path ending by u leads to a state $j \geq i$.*

Lemma 5 *Let $w \in \Sigma^*$ be a word recognized by Oracle(p) in i, then any suffix of w is recognized in a state $j \leq i$.*

The number of states of *Oracle*(p) with $p = p_1p_2 \ldots p_m$ is $m + 1$. We now consider the number of transitions.

Lemma 6 *The number $T_{Or}(p)$ of transitions in Oracle$(p = p_1p_2 \ldots p_m)$ satisfies $m \leq T_{Or}(p) \leq 2m - 1$.*

2.2 On-line Algorithm

This section presents an on-line construction of the automaton $Oracle(p)$, that means a way of building the automaton by reading the letters of p one by one from left to right.

We denote $repet_p(i)$ the longest suffix of $pref_p(i)$ that appears at least twice in $pref_p(i)$.

We define a function S_p defined on the states of the automaton, called supply function, that maps each state $i > 0$ of $Oracle(p)$ to state j in which the reading of $repet_p(i)$ ends. We arbitrarily set $S_p(0) = -1$.
Notes:

- $S_p(i)$ is well defined for every state i of $Oracle(p)$ (Corollary 3).
- For any state i of $Oracle(p)$, $i > S_p(i)$ (lemma 3).

We denote $k_0 = m$, $k_i = S_p(k_{i-1})$ for $i \geq 1$. The sequence of the k_i is finite, strictly decreasing and ends in state 0. We denote

$$CS_p = \{k_0 = m, k_1, \dots, k_t = 0\}$$

the suffix path of p in $Oracle(p)$.

Lemma 7 *Let $k > 0$ be a state of $Oracle(p)$ such that $s = S_p(k)$ is strictly positive. We denote $w_k = repet_p(k)$ and $w_s = repet_p(s)$. Then w_s is a suffix of w_k.*

Corollary 4 *Let $CS_p = \{k_0, k_1 \dots, k_t = 0\}$ be the suffix path of p in $Oracle(p)$ and let $w_i = repet_p(k_{i-1})$ for $1 \leq i \leq t$ and $w_0 = p$. Then, for $0 < l \leq t$, w_l is a suffix of all the w_i, $0 \leq i < l \leq t$.*

We now consider for a word $p = p_1 p_2 \dots p_m$ and a letter $\sigma \in \Sigma$ the construction of $Oracle(p\sigma)$ from $Oracle(p)$.

We denote $Oracle(p) + \sigma$ the automaton $Oracle(p)$ on which a transition by σ from state m to state $m+1$ is added. We already notice that a transition that exists in $Oracle(p) + \sigma$ also exists in $Oracle(p\sigma)$, so that the difference between the two automata may only rely on transitions by σ to state $m+1$ that have to be added to $Oracle(p) + \sigma$ in order to get $Oracle(p\sigma)$.

We are investigating states from which there may be transitions by σ to state $m+1$.

Lemma 8 *Let k be a state of $Oracle(p) + \sigma$ such that there is a transition from k by σ to $m+1$ in $Oracle(p\sigma)$. Then k has to be one of the states on the suffix path $CS_p = \{k_0 = m, k_1, \dots, k_t = 0\}$ in $Oracle(p) + \sigma$.*

Among the states on the suffix path of p, every state that has no transition by σ in $Oracle(p) + \sigma$ must have one in $Oracle(p\sigma)$. More formally, the following lemma sets this fact.

Lemma 9 *Let $k_l < m$ be a state on the suffix path $CS_p = \{k_0 = m, k_1, \dots, k_t = 0\}$ of state m in Oracle$(p = p_1p_2\dots p_m)+\sigma$. If k_l does not have a transition by σ in Oracle(p), then there is a transition by σ from k_l to $m+1$ in Oracle$(p\sigma)$.*

Lemma 10 *Let $k_l < m$ be a state on the suffix path $CS_p = \{k_0 = m, k_1, \dots, k_t = 0\}$ in Oracle$(p = p_1p_2\dots p_m) + \sigma$. If k_l has a transition by σ in Oracle$(p) + \sigma$, then all the states k_i, $0 \leq i \leq t$ also have a transition by σ in Oracle$(p) + \sigma$.*

The idea of the on-line construction algorithm is the following. According to the three lemmas 8, 9, 10, to transform *Oracle*$(p) + \sigma$ in *Oracle*$(p\sigma)$ we only have to go down the suffix path $CS_p = \{k_0 = m, k_1, \dots, k_t = 0\}$ of state m and while the current state k_l does not have an exiting transition by σ, a transition by σ to $m + 1$ should be added (lemma 9). If k_l already has one, the process ends because, according to lemma 10, all the states k_j after k_l on the suffix path already have a transition by σ.

If we only wanted to add a single letter, the preceding algorithm would be enough. But, as we want to be able to build the automaton by adding the letters of p the one after the other, we have to be able to update the supply function $S_{p\sigma}$ of the new automaton *Oracle*$(p\sigma)$. As (according to the definition of S_p), the supply function of states $0 \leq i \leq m$ does not change from *Oracle*(p) to *Oracle*$(p\sigma)$, the only thing to do is to compute $S_{p\sigma}(m + 1)$. This is done with the following lemma.

Lemma 11 *If there is a state k_d which is the greatest element of $CS_p = \{k_0 = m, k_1, \dots, k_t = 0\}$ in Oracle(p) such that there is a transition by σ from k_d to a state s in Oracle(p), then $S_{p\sigma}(m + 1) = s$ in Oracle$(p\sigma)$. Else $S_{p\sigma} = 0$.*

From these lemmas we can now deduce an algorithm **add_letter** to transform *Oracle*(p) in *Oracle*$(p\sigma)$. It is given in Figure 4.

Lemma 12 *The algorithm* **add-letter** *really builds Oracle$(p = p_1p_2\dots p_m\sigma)$ from Oracle$(p = p_1p_2\dots p_m)$ and update the supply function of the new state $m + 1$ of Oracle$(p\sigma)$.*

The complete on-line algorithm to build *Oracle*$(p = p_1p_2\dots p_m)$ just consits in adding the letters p_i one by one from left to right. It is given in Figure 5.

Theorem 1 *The algorithm* Oracle-on-line$(p = p_1p_2\dots p_m)$ *builds Oracle(p).*

Theorem 2 *The complexity of* Oracle-on-line$(p = p_1p_2\dots p_m)$ *is $O(m)$ in time and in space.*

```
**Fonction add_letter**($Oracle(p = p_1 p_2 \dots p_m)$, $\sigma$)
    Create a new state $m + 1$
    Create a new transition from $m$ to $m + 1$ labeled by $\sigma$
    $k \leftarrow S_p(m)$
    **While** $k > -1$ and there is no transition from $k$ by $\sigma$ **Do**
        Create a new transition from $k$ to $m + 1$ by $\sigma$
        $k \leftarrow S_p(k)$
    **End While**
    **If** $(k = -1)$ **Then** $s \leftarrow 0$
    **Else** $s \leftarrow$ where leads the transition from $k$ by $\sigma$.
    $S_{p\sigma}(m + 1) \leftarrow s$
    **Return** $Oracle(p = p_1 p_2 \dots p_m \sigma)$
```

Fig. 4. Add a letter σ to $Oracle(p = p_1 p_2 \dots p_m)$ to get $Oracle(p\sigma)$

```
**Oracle-on-line**($p = p_1 p_2 \dots p_m$)
    Create $Oracle(\epsilon)$ with:
            one single state 0
            $S_\epsilon(0) \leftarrow -1$
    **For** $i \leftarrow 1$ à $m$ **Do**
        $Oracle(p = p_1 p_2 \dots p_i) \leftarrow$ add_letter($Oracle(p = p_1 p_2 \dots p_{i-1})$,$p_i$)
    **End For**
```

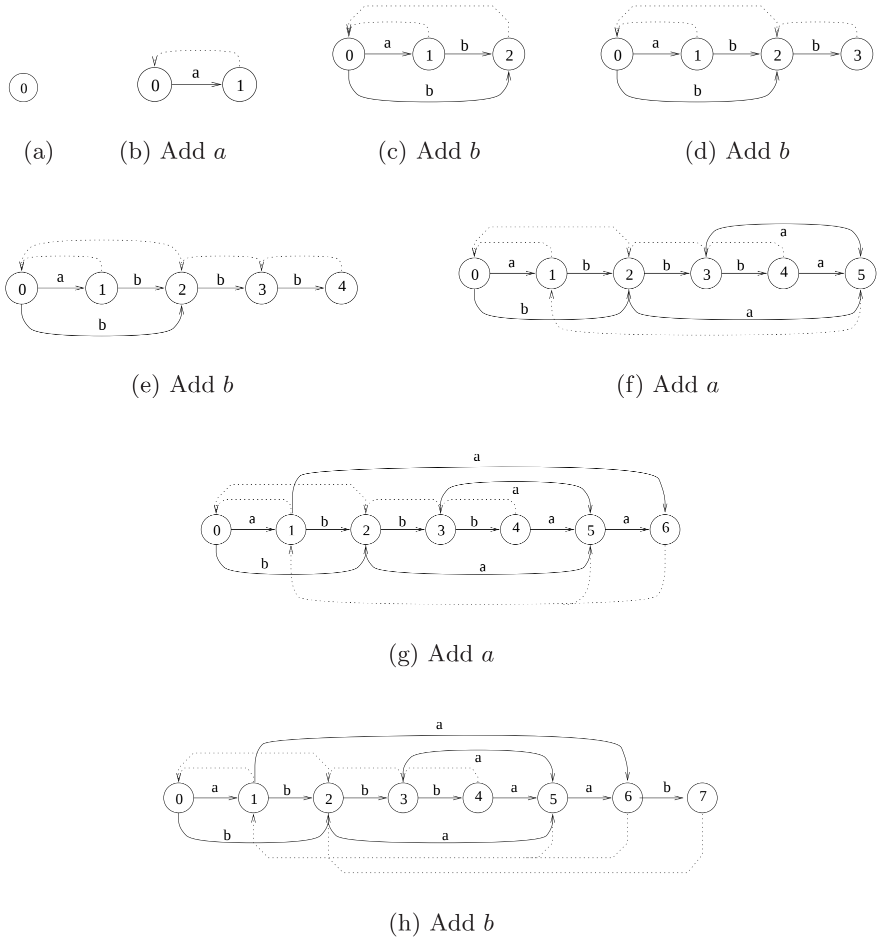
Fig. 5. On-line construction algorithm of $Oracle(p = p_1 p_2 \dots p_m)$

Note The constants involved in the asymptotic bound of the complexity of the on-line construction algorithm depend on the implementation and may involve the size of the alphabet Σ. If we implement the transitions in a way that they are accessible in $O(1)$ (use of tables), then the complexity is $O(m)$ in time and $O(|\Sigma| \cdot m)$ in space. If we implement the transitions in a way that they are accessible in $O(log|\Sigma|)$ (use of search trees), then the complexity is $O(log|\Sigma| \cdot m)$ in time and $O(m)$ in space.

Example The on-line construction of $Oracle(abbbaab)$ is given in Figure 6.

3 String Matching

The factor oracle of p can be used in the same way as the suffix automaton in string matching in order to find the occurrences of a word $p = p_1 p_2 \dots p_m$ in a text $T = t_1 t_2 \dots t_n$ both on an alphabet Σ.


0
0 a 1
0 a 1 b 2 b
0 a 1 b 2 b 3 b
0 a 1 b 2 b 3 b 4 b
0 a 1 b 2 b 3 b 4 a 5 a b a
0 a 1 b 2 b 3 b 4 a 5 a 6 a a b a
0 a 1 b 2 b 3 b 4 a 5 a 6 b 7 a a b a

(a) (b) Add a (c) Add b (d) Add b

(e) Add b (f) Add a

(g) Add a

(h) Add b

Fig. 6. On-line construction of *Oracle*($abbaba$). The dot-lined arrows represent the supply function

The suffix automaton is used in [14,11] to get an optimal algorithm in the average called BDM (for *Backward Dawg matching*). Its average complexity is in $O(n \log_{|\Sigma|}(m)/m)$ under a Bernouilli model of probability where all the letters are equiprobable.

The BDM algorithm move a window of size m on the text. For each new position of this window, the suffix automaton of p^r (the mirror image of p) is used to search for a factor of p from the right to the left of the window.

The basic idea of the BDM is that if this backward search failed on a letter σ after the reading of a word u then σu is not a factor of p and moving the beginning of the window just after σ is secure. This idea is then refined in the BDM using some properties of the suffix automaton.

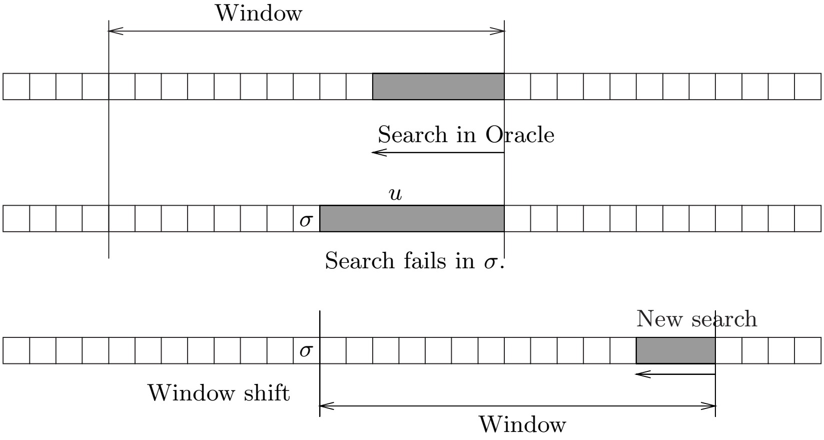


Fig. 7. Shift of the search window after the fail of the search by *Oracle*(p). The word σu is not a factor of p

However this idea is enough in order to get an efficient string matching algorithm. The most amazing is that the strict recognition of the factors (that the factor and suffix automata allow) is not necessary. For the algorithm to work, it is enough to know that $u\sigma$ is not a factor of p. The oracle can be used to replace the suffix automaton as it is illustrated by Figure 7. We call this new algorithm **BOM** for *Backward Oracle Matching.* The pseudo-code of BOM is given in Figure 3. Its proof is given lemma 13. We make the conjecture (according to the experimental results) that BOM is still optimal in the average.

Lemma 13 *The BOM algorithm marks all the occurrences of p in T and only them.*

The worst-case complexity of BOM is $O(nm)$. However, in the average, we make the following conjecture based on experimental results (see 3.2):

Conjecture 1 *Under a model of independance and equiprobability of letters, the BOM algorithm has an average complexity of $O(n \log_{|\Sigma|}(m)/m)$.*

3.1 A Linear Algorithm in the Worst Case

Even if the preceding algorithms are very efficient in practice, they have a worst-case complexity in $O(mn)$. There are several techniques to make the BDM algorithm (using suffix automaton) linear in the worst case, and one of them can also be used to make our algorithms linear in the worst case. It uses the Knuth-Morris-Pratt (KMP) algorithm to make a forward reading of some characters in the text.

```
BOM($p = p_1p_2 \ldots p_m$, $T = t_1t_2 \ldots t_n$)
    Pre-processing
        Construction of the oracle of $p^r$
    Search
        $pos \leftarrow 0$
        While ($pos <= n - m$) do
            $state \leftarrow$ initial state of $Oracle(p^r)$
            $j \leftarrow m$
            While $state$ exists do
                $state \leftarrow$ image state by $T[pos + j]$ in $Oracle(p^r)$
                $j \leftarrow j - 1$
            EndWhile
            If $j = 0$ do
                mark an occurrence at $pos + 1$
                $j \leftarrow 1$
            EndIf
            $pos \leftarrow pos + j$
        EndWhile
```

Fig. 8. Pseudo-code of **BOM** algorithm

To explain the combined use of KMP and (factor or suffix) oracle, we consider the current position before the search with the oracle: a prefix v of the pattern has already be read with KMP at the beginning of the search window and we start the backward search using the oracle from the right end of that current window. The end position of v in the current window is called *critical position* and is denoted by *Critpos*. The current position is schematized in Figure 9.

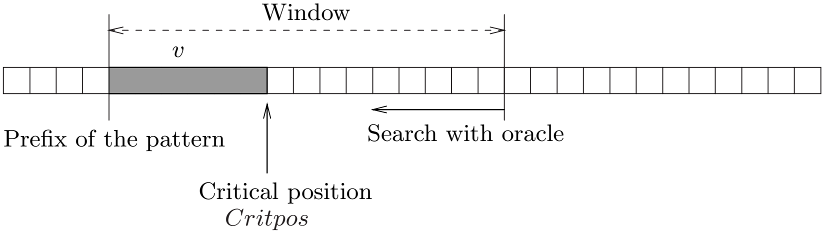


Fig. 9. Current position in the linear algorithm using both KMP and (factor or suffix) oracle

We use the search with the oracle from right to left from the right end of the window. We consider two cases whether the critical position is reached or not.

1. The critical position is not reached. The failure of the recognition of a factor occurs on character σ as in the general approach (Figure 7). We shift the window to the left until its beginning goes past character σ. We restart a KMP search on this new window rereading the characters already read by the oracle. This search stops in a new current position (with a new corresponding critical position) when the recognized prefix is small enough (less than αm with $0 < \alpha < 1$). The value of α is discussed with the experimental results (see Section 3.2), typically $\alpha = 1/2$. This situation is schematized Figure 10.

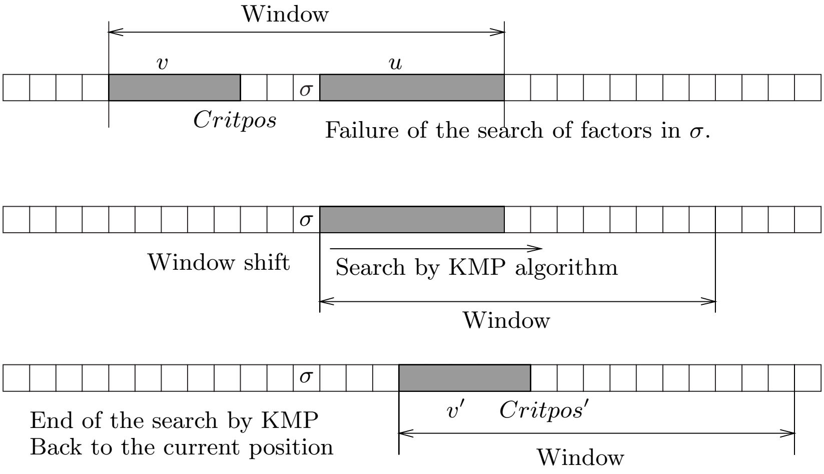


Fig. 10. First case: the critical position is not reached

2. The critical position is reached. We resume the KMP search from the critical position, from the state we were before stopping, rereading at least the characters read by the oracle. We then go on reading the text until the longest recognized prefix is small enough (less than α). This situation is schematized Figure 11.

This algorithm can be used with a backward search done with the factor oracle. We call this new algorithm **Turbo-BOM**. Concerning the complexity in the worst case, we have the following result.

Theorem 3 *The algorithm Turbo-BOM is*

(i) linear considering the number of inspections of characters in the text. The number of these inspections is less than $2n$*.*

(ii) linear considering the number of comparisons of characters. The number of these comparisons is less than $2n$ *when the transitions of the oracle are available in* $O(1)$ *and less than* $2n + n \log \Sigma$ *when the transitions are available in* $\log \Sigma$*.*

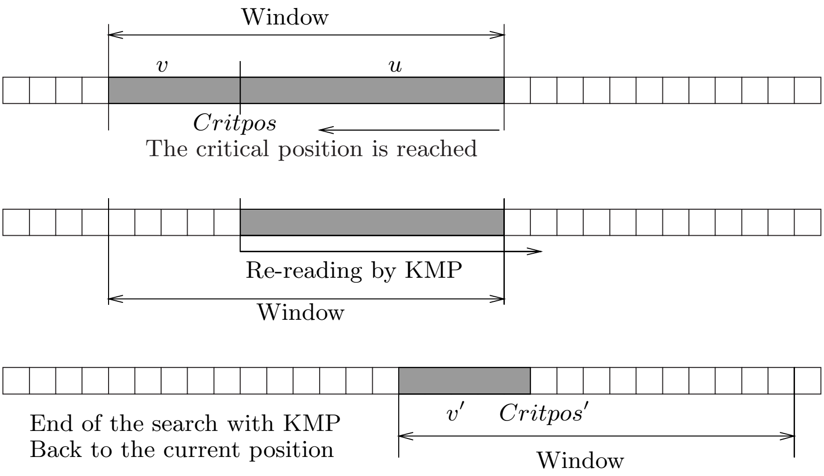

Window
v
u
$Critpos$
The critical position is reached
Re-reading by KMP
Window
End of the search with KMP
Back to the current position
v'
$Critpos'$
Window

Fig. 11. Second case: the critical position is reached

3.2 Experimental Results

In this section, we present the experimental results obtained. More precisely, we compare the following algorithms.

- **Sunday**: the Sunday algorithm [15] is often considered as the fastest in practice,
- **BM**: the Boyer-Moore algorithm [6],
- **BDM**: the classical Backward Dawg Matching with a suffix automaton [11],
- **Suff**: the Backward Dawg Matching with a suffix automaton but without testing terminal states, this is equivalent to the basic approach with the factor automaton[2],
- **BOM**: the Backward Oracle Matching with the factor oracle,
- **BSOM**: the Backward Oracle Matching with the suffix oracle. This later structure is not described in this version of the paper, but can be found in [1].
- **Turbo-BOM**: the linear algorithm using BOM and KMP with $\alpha = 1/2$.

Our string matching experiments are done on random texts of size 10 Mb with an accuracy of ±2 % with a confidence of 95 % (which may require thousands of iterations) for alphabets of size 2, 4, 16 and 32. The machine used is a PC with a

[2] The suffix automaton without taking in account the terminal states (i.e. considering every state as terminal) and the factor automaton recognize the same language. The difference is that the factor automaton is minimal, so its size is smaller or equal than the size of the suffix automaton. But the difference of size is not significant in practice, anyway not enough significant to justify the implementation of a factor automaton which will complicate and slow the preprocessing phase of the string matching algorithm.

Pentium II processor at 350 MHz running Linux 2.0.32 operating system. For all the algorithms, the transitions of the automata are implemented as tables which allow $O(1)$ branches. But it is not realistic (especially for the suffix automaton) when the alphabet becomes rather big (for instance for 16 bits character coding). Moreover, the Sunday algorithm becomes unusable as it is when the alphabet is big because it mainly uses character table.

Experimental results in string matching are always surprising because codes are small and the time taken by a comparison is not much greater than the time taken by an indice incrementation. It is for instance the reason why Sunday algorithm (when it is usable) is the fastest algorithm for small patterns. The window shift are very small but very few operations are necessary to get this shift. It is also the reason why BDM is slower than Suff whereas the window shifts in BSOM and BDM are greater.

The 4 subfigures of Figure 12 shows that BOM is as fast as Suff (except on a binary alphabet) which is much more complicated and requires much more memory.

It is obviously useless (in the case of searchs in texts of characters) to mark and test terminal states in both suffix automaton and factor oracle.

Turbo-BOM algorithm is the slowest but it is the only one that can be used in real time and in that case its behavior is rather good. It has to be noticed that we arbitrarily set the value of α to $1/2$. However, according to the tests we have proceeded for different values of α, it turns out that $\alpha = 1/2$ is the more often the best value and that the variations of search times with other values of α (as far as they stay between $(2\log_{|\Sigma|} m)/m$ and $(m - 2\log_{|\Sigma|} m)/m$) are not very significant and anyway do not deserve by themselves an accurate study.

4 Conclusions

The new structure we presented, the *factor oracle*, allows new string matching algorithms. These algorithms are very efficient in practice, as efficient as the ones which already exists, but are far more simple to implement and require less memory. According to the experimental results, we conjecture that they are optimal on the average (under a model of equiprobability of letters) but it remains to be shown.

About the structure of factor oracle itself, many questions stay open. Among others, it would be interesting to have a characterization of the language recognized by the oracle.

It would also be interesting to have a study of the average number of external transitions in the oracle. It would give an idea of the average memory space required by the string matching algorithms.

Finally, we notice that the factor oracle is not minimal considering the number of transitions among the automata of $m + 1$ states which recognize at least the factors. An example is given in Figure 13. This reduced automaton may also be used in string matching provided that its construction can be done in linear time. This construction remains an open problem.

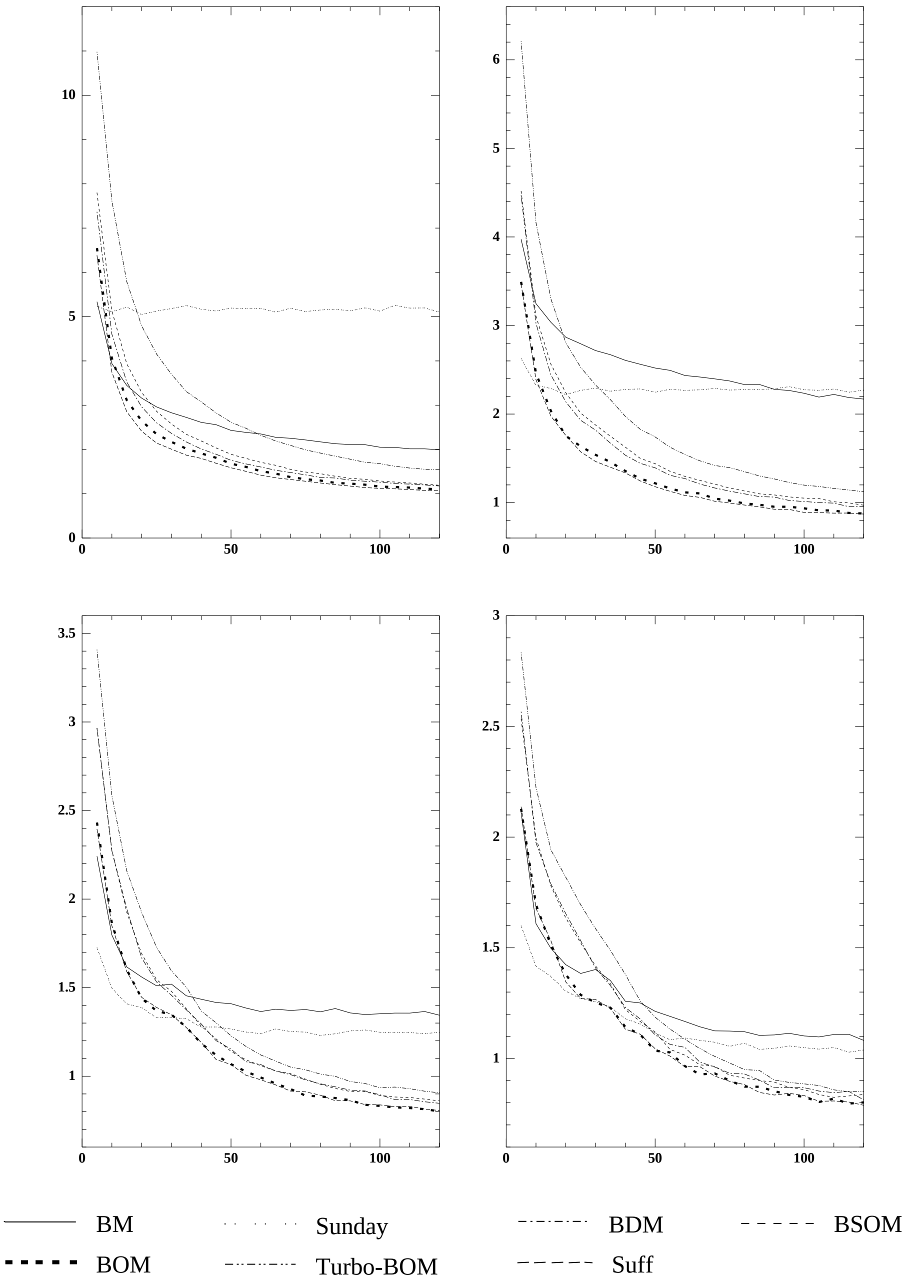
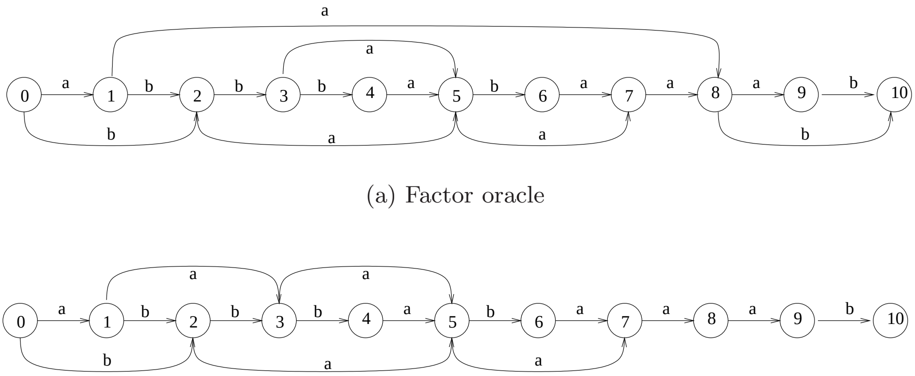


Fig. 12. Experimental results in time of the string matching algorithms on random texts of size 10 Mb on alphabets of size 2, 4, 16 and 32. The X-axis represents the length of the pattern and the Y-axis the search time in 1/100th seconds per Mbyte



(a) Factor oracle

(b) Reduced automaton

Fig. 13. The factor oracle is not minimal considering the number of transitions among the automata of $m + 1$ states which recognize at least the factors

References

1. C. Allauzen, M. Crochemore, and M. Raffinot. Factor oracle, Suffix oracle. Technical Report 99-08, Institut Gaspard-Monge, Université de Marne-la-Vallée, 1999. http://www-igm.univ-mlv.fr/ raffinot/ftp/IGM99-08-english.ps.gz. 296, 306
2. C. Allauzen and M. Raffinot. Oracle des facteurs d'un ensemble de mots. Rapport technique 99-11, Institut Gaspard Monge, Université de Marne-la-Vallée, 1999. http://www-igm.univ-mlv.fr/ raffinot/ftp/IGM99-11.ps.gz. 296
3. R. A. Baeza-Yates. Searching subsequences. *Theor. Comput. Sci.*, 78(2):363–376, 1991. 296
4. A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, M. T. Chen, and J. Seiferas. The smallest automaton recognizing the subwords of a text. *Theor. Comput. Sci.*, 40(1):31–55, 1985. 296
5. A. Blumer, A. Ehrenfeucht, and D. Haussler. Average size of suffix trees and DAWGS. *Discret. Appl. Math.*, 24:37–45, 1989. 296
6. R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977. 306
7. M. Crochemore. Transducers and repetitions. *Theor. Comput. Sci.*, 45(1):63–86, 1986. 296
8. M. Crochemore, F. Mignosi, and A. Restivo. Automata and forbidden words. *Information Processing Letters*, 67(3):111–117, 1998. 296, 309
9. M. Crochemore, F. Mignosi, and A. Restivo. Minimal forbidden words and factor automata. In L. Brim, J. Gruska, and J. Zlatuška, editors, *Mathematical Foundations of Computer Science 1998*, number 1450 in LNCS, pages 665–673. Springer-Verlag, 1998. Extended abstract of [8]. 296
10. M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi. Text compression using antidictonaries. Rapport I.G.M. 98-10, Université de Marne-la-Vallée, 1998. 296

11. M. Crochemore and W. Rytter. *Text algorithms*. Oxford University Press, 1994. 295, 302, 306
12. M. Crochemore and R. Vérin. Direct construction of compact directed acyclic word graphs. In A Apostolico and J. Hein, editors, *Combinatorial Pattern Matching*, number 1264 in LNCS, pages 116–129. Springer-Verlag, 1997. 296
13. M. Crochemore and R. Vérin. On compact directed acyclic word graphs. In J. Mycielski, G. Rozenberg, and A. Salomaa, editors, *Structures in Logic and Computer Science*, number 1261 in LNCS, pages 192–211. Springer-Verlag, 1997. 296
14. A. Czumaj, M. Crochemore, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter. Speeding up two string-matching algorithms. *Algorithmica*, 12:247–267, 1994. 302
15. D. Sunday. A very fast substring search algorithm. *CACM*, 33(8):132–142, August 1990. 306

Principles of Forecasting – A Short Overview

Emil Pelikán

Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Praha 8, Czech Republic
emil@uivt.cas.cz

Abstract. Forecasting procedures are needed when there is uncertainty about the future. In our contribution we discuss some principles that can help to make more accurate forecasts and help to better assess the uncertainty associated with forecasts. We mainly discuss statistical forecasting procedures, but other principles based on experts (judgmental forecasting) and integrating and combining approaches are also mentioned. We show some results of forecasting in two different application areas.

1 History

Forecasting represents an important phenomenon for all individual beings, communities and societies. Importance of forecasting of the future has been recognized since very ancient times. Antique prophets were celebrated for their successful predictions and were cursed when their forecasts failed. During the centuries different forecasting methods have been developed, improved and applied. For example, Ptolemaios concept of the Universe was developed 1900 years ago and astronomers could predict the most significant sights in the sky. Later, the systematic errors were identified by astronomers using many and many observations. The Copernican concept of the Universe represented the significant innovation allowing astronomers to predict the movement of the stars with fascinating accuracy. Modern astronomy is, of course, more accurate than Copernican astronomy. Similar progress can be monitored in the theory of motion with Aristotle, Galileo, Newton and Einstein concepts.

Forecasting during the first six decades of this century were oriented to "judgmental" or "by hand" forecasts, numerical calculations were difficult and extremely time-consuming, and practical applicability of such approaches were limited to simple time series. Nevertheless, many useful forecasting knowledge has been gained. Gordon [23] found that the average judgment from groups of human forecasters was substantially more accurate than those from the typical individual expert. Ogburn [33] and MacGregor [30] found that judgmental forecasts were strongly influenced by the desired outcome (optimism bias). Some experts concluded [18] that forecasters should use as long a time series as possible. The principle sometimes conflicts with the principle of using the most relevant data, which typically means the most recent data. Jarvik [26] identified the "gambler's fallacy", whereby people expect that earlier outcomes will affect the next one in games of chance. This phenomenon can be observed in many

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 311–327, 1999.

other areas of human behavior. In Brown's [9] extrapolation model, the historical data were weighted with exponentially decreasing weights. This produces more accurate prediction for rapidly changing (nonstationary) data. Ferber [19] declared that the fit error of a model was a poor criterion for its forecast abilities. This was a new message for many statisticians in that time. Theil [52] developed useful measures for assessing time series forecast errors. His U-statistics allows a relative comparison of different forecasting methods.

Since 1960 the wide advent of computing machines has opened a much wider area for forecasting methods. The increasing computing power enables not only to quantitative extension of previously known principles, but principally new approaches were developed and applied.

2 Why Forecast?

Regadless of the progress in forecasting methods during last decades, two important point should be mentioned. The first point is that sucessful forecasting is not always directly useful to managers and other users. (Jules Verne correctly predicted submarines, travel to the moon etc., but there were no help to engineers, how to construct such inventions). The second point is that the forecasting procedures are needed only if there is uncertainty about the future. There is no uncertainty when one can control events. For example, we do not need to predict the temperature in the refrigerator. Forecasting itself is only one step (following the data collecting, archiving and checking). The forecasting results form the inputs for the subsequent planning and decision making step. Schematically, three steps of the whole process are shown in Fig. 1. Of course, some more complicated scheme with feedback connections can be taken into account. If the outputs are not satisfactory, the plans can be revised. This process can be repeated until forecasts are satisfactory. Revised plans are then implemented and actual outputs are monitored for use in the next planning period.

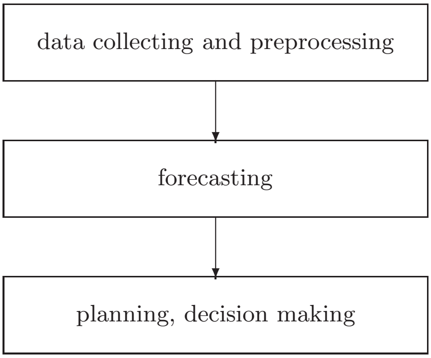


Fig. 1. Forecasting role in the planning and decision making process

It is now necessary to emphasize the difference between forecasting and planning. Whereas planning is concerned with what the world *should* look like, forecasting is concerned with what it *will* look like. However, in practice, many organizations revise the forecasts, not the plans. This is done with the belief that forecasts affect behavior. Sometimes it can be true. For example, the expected exchange rate between USD and EURO (or CZK) can affect behaviour of dealers.

3 Economical and Legal Aspects of Forecasting

The forecasting procedures are very useful in many areas of human activities including medicine, economy, energy industry, banking, telecommunications, transportation, environment etc. Organizations invest enormous amount of money on forecasts about new products, factories, retail outlets etc. The companies expect that their investment into more accurate forecasting technologies will bring more profit in the future. In many cases a lot of money can be save when forecasters improve their predictions. It has been published in [10], that for electric companies in the UK, an incerase of 1 % in forecasting error of the electric load would be associated with an increase in operational costs about 10 millions British pounds per year (prices from 1984). What is happenned, if the forecast is wrong and economic or other losses are enormous? Here are some cases cited in [5]:

> *"Four Massachusetts fishermen were lost at sea on November 21, 1980 because, their families claimed, of an incorrect weather forecast. Three families brought suit and won an initial judgment on the ground that the National Weather Service was negligent in failing to repair a weather buoy that might have provided useful data. ... ... The key issue in this case was not the fact that the forecast was wrong, but whether the National Oceanic and Atmospheric Administration (NOAA) failed to take reasonable steps to obtain accurate data. Another issue was that when key information was no longer available, NOAA did not notify the users of the forecast. ... ... "*

> *"In a British case, Esso Petroleum vs. Mardon (London, 1966 E. no. 2571), Mardon entered into a contract with Esso to own and operate a gas station. A critical part of the negotiations was the forecast that the station would sell 200,000 gallons of gas per year by the third year. The actual sales fell well short of the forecasted figure, and Mardon went out of business. Esso sued Mardon for unpaid bills. Mardon then countersued on the basis that the Esso forecast misrepresented the situation. In effect, Esso had originally forecast the 200,000 gallon figure under the assumption that the gas pumps would face the road. After a zoning hearing, they were forced to change the design so that the pumps were not visible from the road. Despite this significant unfavorable change, Esso then used the 200,000 gallon forecast in the original contract with Mardon. Mardon won; the court concluded that Esso had misrepresented the situation."*

> *"In Beecham vs. Yankelovich, Beecham alleged that an inaccurate market forecast prepared by Yankelovich Clancy Shulman resulted in a $ 24 million loss. Yankelovich, on the other hand, claimed that Beecham provided incorrect inputs to the forecasting models, and that they failed to follow the marketing plan; for example, they changed advertising claims and reduced promotional expenses (Adweek's Marketing Week, 7 December 1987, pp. 1,4)."*

These cases imply that if there is no special contract between the forecaster and the user, forecaster are unlikely to be held liable. In all cases forecasts contain uncertainty, and reasonable balance between loses and benefits should be taken into account. Forecasters can be held liable if it can be shown that the forecasts were not obtained by reasonable practice, and only if the poor practice influences the results.

4 The Forecasting Process

There are many forecasting methods and the application of the concrete one depends upon the situation. For example, for long-range forecasting of the market, econometric methods are often appropriate. For short-range forecasting of costs, sales, or market share, extrapolation methods are useful. There is *no universal (the best) forecasting method*, which can be applied in all areas and for all prediction horizons. In this part we show general aspects of the forecasting methodology which can lead to the development of the "optimal" prediction models. Schematically, the whole forecasting process is shown in Fig. 2.

4.1 Problem Identification and Description

The first step in forecasting is to specify the prediction problem. The definition of the problem is sometimes the most difficult aspect. It is necessary to deep understand how the forecast will be used, who requires the forecasts, which variables will be predicted, how long the forecasting horizon is, which data is available in the proper (e.g. digital) form, what is the nature of processed data and how it might be transformed. It is worth to spend some time talking to experts who are involving in collecting data, forming the database, using the forecasts for the planning and decision making process. Forecasters should know as much as possible about the whole concept of the building systems.

4.2 Relevant Input Variables Selection

This aspect of the forecasting methodology is very important and the proper selection of suitable input variables has a great effect on the final prediction results. Correlation analysis can be used when linear models are good candidates. But, the variable selection for nonlinear models is very difficult. Correlation analysis cannot be used, because it only detects linear dependencies.

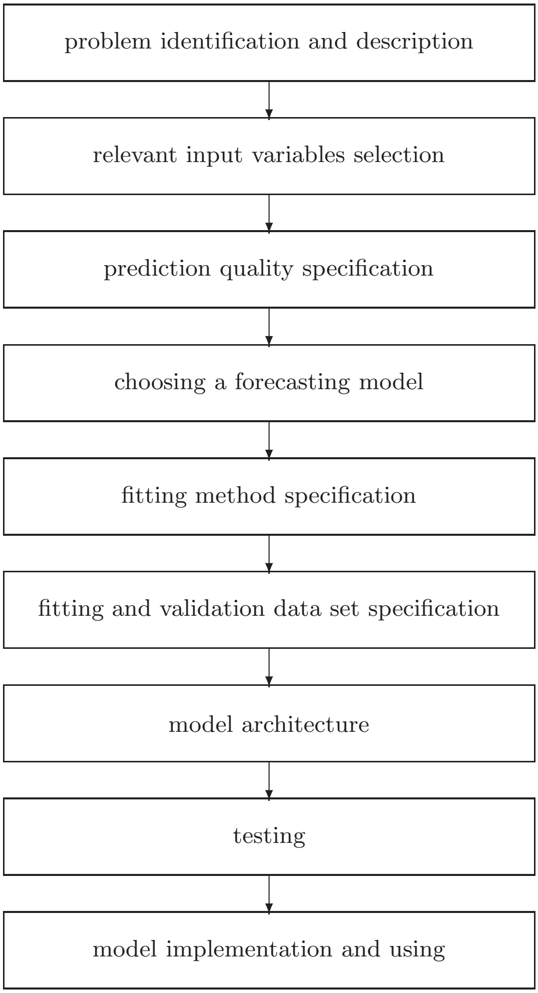

problem identification and description
relevant input variables selection
prediction quality specification
choosing a forecasting model
fitting method specification
fitting and validation data set specification
model architecture
testing
model implementation and using

Fig. 2. Aspects of the forecasting methodology

One approach is to apply the statistical tests for detecting the nonlinearities. Some tests can be found in [54] and [53]. Other interesting approaches, based on mutual information between two and more variables, are presented in [16] and [37]. These methods can help in proper variable selection as well as helping in decision, if an linear or nonlinear model is necessary. Another usefull approach is based on a heuristic selection of possible input variables and on the comparison of results from the different models. Any apriori information (judgmental or received from key persons) is valuable during this step (e.g. we apriori know that outdoor temperatures influence the electric consumption). This heuristic approach can be combined with some searching techniques, as are, for example, genetic algorithms [34].

4.3 Prediction Quality Specification

Because there is not a single universally accepted measure of accuracy in time series forecasting, the aspect of the prediction error specification should be taken into account very carefully. The mean squared error function (MSE) or mean absolute percentage error function (MAPE) are the most popular criteria for the prediction quality evaluation. In some cases the maximal errors or the whole prediction error distribution plays a more significant role than the usual MSE or MAPE values. Theil's U-statistics, the Durbin-Watson statistics or some other statistics can also be very useful measures (see [31]). The traditional fitting techniques are based on minimizing of the MSE function for data from the "training set". Some other fitting methods, which penalized the higher prediction errors, can be applied to reduce the fatal forecasts.

4.4 Choosing a Forecasting Model

There is no universal forecasting method. Each model is based on a set of assumptions and it involves one or more parameters which must be estimated using the known data sets. Selection must be made considering the characteristics of the data and the frequency of required forecasts. For example, if we need to forecast hundreds items on daily basis, there is no time to interact with prediction software to built new models. A simple and reliable automatic prediction system should be implemented in such case. There is a family of statistical parametric models, which try to model a relationship between the predicted values and the inputs from the past data. This family includes decomposition models, exponential smoothing models, regression models, Box-Jenkins ARIMA models, regression models with ARIMA errors, artificial neural networks, intervention models, state space models and others. If data include trend, seasonality or other "regular" components, the decomposition model help us to estimate these parts. If there are strong correlations in the data, ARIMA model can successfully detect the linear relationships. Nonlinear dependencies can be modelled by artificial neural networks. Exponential smoothing can be applied for short-term extrapolations of "short" time series. Selection of a suitable prediction model is still in the "art" domain.

4.5 Fitting Method Specification

Ordinary least square, maximum likelihood and other approaches have been developed for estimating (fitting) parameters of the models from available data sets. Many other optimizing methods have been used for minimizing selected error functions (sometimes with penalty term) with respect to the model parameters. The most popular metod for neural networks is the backpropagation algorithm which is a gradient steepest descent method. A number of modifications of this algorithm have been proposed. Among them, the second order methods such as BFGS and Levenberg-Marquardt methods are more efficient and can significantly reduce the fitting time and improve the prediction accuracy. In the case a of non-continuous error function some other methods should be selected (e.g based on stochastic gradient methods).

4.6 Fitting and Validation Data Specification

The suitability of a particular forecasting method is measured by a selected error function. This function should not reflect the "goodness-of-fit", but it must reflect an ability to perform the forecast in the future. So, the available data are split into the fitting and validation sets. It is crucial to have both parts as representatives. An inappropriate separation will affect the forecasting performance. Most authors select a specific part of the available data (e.g. 70 %) for fitting and the rest for the validation process. Other authors employ a bootstrap technique [24]. Another factor is the sample size. If we have huge data sets, which part could be used of optimal model fitting? If there not enough data, which model can or cannot be estimated? No exact rule exists for the determination of the sample size for a given problem. But a larger sample can be required when a more complex relationship is modelled or the noise in the data increases. The selection of the sample size is also related to number of parameters. A heuristic rule often cited is: a sample size should be 10 times greater then the number of parameters in the model [55].

4.7 Model Architecture Optimizing

The aim of the architecture optimizing process is to built the systems with better forecasting ability. The over-parametrized models can be reduced by special prunning techniques. Sensitivity and tolerance analysis can optimize the model structure respecting the principle of simplicity. This principle (principle of parsimony or Occam's Razor principle) says that as few parameters as possible should be used in fitting a model to a set of data. The second approach of this optimizing step is based on combining predictions. When we have more than one forecasts, we can combine them to obtain another one. Experience shows, that, in the majority cases, such combining results in more accurate "post-sample" prediction than we obtained from the individuals.

4.8 Testing

When the forecasting models have been developed using available data sets, it is necessary to test them on new, previously unknown data. This process should be done before final implementation of the forecasting models. During this step, some "wrong" reactions on the "unexpected" events can be tuned. For example, missing data and outliers cause troubles. The comparison with expert judgments yields a good idea about the usefulness of our forecasting systems. The question when our system should be "retrained" should be also answered. In addition, the forecasting accuracy is not the only criterion for evaluation of our models. If the managers, using our forecasts, change their scenario, then the predictions will not be true.

4.9 Model Implementation and Using

Implementing forecast models enabling to yield new information (reliable and in-time) to the management is often at least as important as the forecasts themselves. Forecasters should provide summary of the forecasting methods and supporting data in a simple and understandable form. It is very valuable if the prediction intervals are presented together with the point forecasts. The forecasting system should enable user-friendly on-line connections to data sources, data editing, preprocessing, parameter setting, forecasting, graphical representations and statistical evaluations.

5 Forecasting Formulas

The k-step ahead forecast of the future M-dimensional time complex time series can be realized by a set of functions

$$F(X, P, t+k) = \{F_{1,k}(X_1, P_1,), F_{2,k}(X_2, P_2), \dots , F_{M,k}(X_M, P_M)\} \tag{1}$$

where $F_{j,k}$, $j = 1, \dots , M$ are the functions forecasting the marginal variables. The vector X_j represents the input variables available at time t and P_j is a vector of parameters. For one-dimensional case and one-step ahead forecast of the time series $Y(t)$, the previous formula can be reduced to

$$\hat{Y}(t+1) = F(X, P). \tag{2}$$

Input variables can include the external (independent) variables as well as the internal inputs (the past values of Y, other unobservable components as are the noise terms in ARMA models, etc.). If the function F is linear with respect to a set of inputs $X = \{x_1, x_2, \dots , x_n\}$, then the forecast is realized by the linear regression formula

$$\hat{Y}(t+1) = a_1x_1 + a_2x_2 + \dots + a_nx_n + c, \tag{3}$$

where $P = \{a_1, a_2, \dots a_n, c\}$. If $x_1 = t, x_2 = t^2, \dots , x_n = t^n$, then the forecast is realized by the polynomial regression

$$\hat{Y}(t+1) = a_1t + a_2t^2 + \dots + a_nt^n + c. \tag{4}$$

When we have $x_1 = Y(t), x_2 = Y(t-1), \dots , x_n = Y(t-n+1)$, the forecast is realized by the autoregressive (AR) model

$$\hat{Y}(t+1) = a_1Y(t) + a_2Y(t-1) + \dots + a_nY(t-n+1) + c. \tag{5}$$

Parameters also could be changed, so we have a adaptive AR forecast with time-dependent (usually slowly varied) parameters

$$\hat{Y}(t+1) = a_1(t)Y(t) + a_2(t)Y(t-1) + \cdots + a_n(t)Y(t-n+1) + c(t). \tag{6}$$

If we have $X = \{\hat{Y}(t), Y(t)\}$, then a single exponential smoothing forecast is given by

$$\hat{Y}(t+1) = \hat{Y}(t) + \alpha(\hat{Y}(t) - Y(t)) \tag{7}$$

where α is a parameter from the interval $(0, 1)$. This formula is equivalent to the autoregresive forecast with infinite number of exponentially decreasing parameters.

For nonlinear case, the neural networks (three layered perceptron) can model the relationship by the function

$$F(X) = f\left(\sum_{i=0}^{l} w_i f_i\left(\sum_{j=1}^{n} v_j x_j\right)\right) \tag{8}$$

where f and f_i are the sigmoidal functions of the form

$$f(t) = \frac{a}{1+e^{-bt}} \tag{9}$$

where a, b, w_i, v_j are parameters and x_j are components of the input vector X.

Of course, many other formulas can be derived. Some models are theoretically equivalent (or asymptotically equivalent) under some reasonable conditions. For example, it has been proved that any continuous function can be approximated by the three layered perceptron. But, it would be queer to apply neural networks for modelling of the polynomial regression, which is generally nonlinear, but it is linear in parameters. Another example is that the single exponential smoothing model with one parameter can be expressed as the AR process with infinite order (infinite number of parameters). Estimation of one parameter only is much more pretty procedure.

6 Prediction Intervals

In many application areas it is valuable to provide not only a point forecast but also to some limits for possible "worst" (or best) values. It can be done in the form of the prediction interval (or sets of forecast limits). The deriving of such interval is based on statistical theory and probability distributions. The forecasted limits are related with some percentage level, e.g. 95 %. This means that the prediction interval will contain the observation with probability 95 %.

The prediction interval can be analytically derived for some models (for normally distributed errors). If the forecast errors are normally distributed with zero mean (unbiased forecast) then the 95 % prediction interval can be estimated by

$$\hat{Y}(t+1) \pm 1.96\sqrt{MSE} \tag{10}$$

where MSE means the mean square of the forecast errors. The similar formula can be derived for a k-step ahead forecasting. Sometimes these estimations produce very wide intervals which would be of little practical use.

7 Judgmental Forecasting

The statistical forecasting methods try to find and extrapolate existing relationships and they suppose that such relationships will not dramatically change in the future. (Slow changes can be modelled by adaptive statistical models). Changes should be detected as soon as possible to reduce forecasting errors. Human judgement is needed to correct the forecasting using the "inside information" and his knowledge. However, there are some limits and considerable biases using the judgmental forecasts. *People do not want to be held responsible if their forecast is wrong.* The cost of the human forecast is significantly higher. There is strong inconsistency in human forecasting (people change their mind when there is no need to do, they are influenced by their mood, by the opinion of their colleagues or wife, might be bored). Therefore many companies combine the statistical forecasting with judgmental forecasts to improve the inputs for their decision making process.

8 Forecasting Software

There exists a lot of number of different software forecasting packages which are available on the market. Some companies (e.g. Siemens, Honeywell etc.), produce complete "tailored" solutions and implement the proper forecasting models into the client information system. Of course, there are many other forecasting software and the selection of a suitable one can be difficult. Some small specialized packages contains forecasting instruments which are not implemented in the large general statistical software. Some packages are easy to use and can be operated by users, other software can be applied with experts only. The ability to process large data sets with many variables is another criterion for selection of the suitable package. The most popular statistics packages are SAS with ETS component, SPSS, S-Plus, MiniTab. The examples of the specialized software are Forecast Pro, Autobox, NeuralWorks/Predict, SIBYL/Runner. There are some add-on forecasting function implemented in spreadsheet packages such as Excel, Lotus 1-2-3 and Quattro Pro.

9 Forecasting Journals, Conferences and Internet Addresses

Besides of the journals in statistics, economics, environment and in other fields, which often include articles on forecasting, there are three specialized journals devoted to the forecasting problems. The *"International Journal on Forecasting"* and the *"Journal on Forecasting"* are the leading journals in the field. They published theoretically and practically oriented papers covered all areas of forecasting. The *Journal of Business Forecasting* includes mainly papers with practical aspects of forecasting.

There are two forecasting associations. *The International Institute of Forecasters (IIF)* is an association that includes both academics and practitioners. It hosts the annual conference International symposium of Forecasting (ISF). *The International Association of Business Forecasters (IABF)* includes mostly people interesting in business forecasting. These two associations published the joint newsletter *(The Forum)* which includes information about the forecasting conferences, reviews of forecasting software, short articles and news about forecasting.

The information presented above together with FAQ's (frequently asked questions), time series data, links to other resources and more can be find at the IIF home page

http://forecasting.cwru.edu.

Since 1998, the prediction conferences NOSTRADAMUS have been organized in the Czech Republic. Further information can be found at

http://ft3.zlin.vutbr.cz/NOSTRA/NOSTRA.htm.

10 Two Examples

10.1 Foreign Exchange Rate Forecasting

The foreign exchange rate is very complex time series which is influenced by many factors. We tested to predict of high-frequency foreign exchange quotes of the USD/DEM currency collected by the *Olsen & Associates* in the period 1992-1993. We predicted the median of the quotes calculated by the moving window technique with size of 50 quotes. We had no "inside" information about the time series, so we decided to use the statistical methods for finding the relationships between the past data and the predicted quantities. The average of the prediction horizon (APH) was about 18 minutes, so we tried to predict the exchange rates for next 18 minutes in average. I asked three of mine colleagues (Dr. Ladislav Pecen, Dr. Petr Klán and Dr. Petr Berka) to develop independently their models using the data from six months (November, 1992 – April, 1993). Five statistical methods were developed my me and by my colleagues and we tested their performance in May, 1993. We applied the stochastic differential equation based predictor ST and smoothing spline predictor SP (developed by

L. Pecen), the convex combination based predictor CC (developed by P. Klán), the neural network model NN (developed by E. Pelikán) and rule-based predictor KEX (developed by P. Berka). The results of our experiment are summarized in Table 1. *Comb* means the strategy, when the prediction is realized only in the case, when all five predictors produce the same results (next trend up or next trend down), *trend* represents the trend agreement score, *ts* means the trading score representing the averaged theoretical profit per one transaction (without transaction costs), *Num* means how many times the predictor forecasts during the given period. All results are compared with the "ideal" solution. We can see, that all methods produced better results than the "tossing up a coin" strategy (trend scores are above the 50 %). The combining strategy improved the prediction performance because the prediction models were developed independently using different principles and different sets of past values.

Table 1. The forecasting performance of five different methods and their combinations using the USD/DEM foreign exchange rate time series (18 minutes ahead forecasts in average)

du50 APH (min): 18 Period: May 93

method	**ST**	**SP**	**CC**	**NN**	**KEX**	**Comb**	**ideal**
trend (%)	52.3	56.4	59.1	58.8	52.5	61.5	100
ts	0.08	0.578	1.09	1.187	0.561	1.777	5.347
num.	2285	2285	2285	2285	2285	413	2285

10.2 Electric Load Forecasting

In the opposite to the previous case, we have much more "inside" information about the electric load time series. We spent a lot of time together with experts from power distribution companies who adviced us what possible dependencies could exist between the electric load variables and the outside temperature, luminosity, type of day, wind speed etc. Therefore we could build the explanatory forecasting models for several days ahead prediction horizon. We used, for example, a sigmoidal function for modelling of the relationship between the electric load values and the daily averaged temperatures. It was inspired by the fact that the electric load increases with decreasing temperatures up to some limit. A saturation effect appears at very low as well as at very high temperatures. Above (resp. below) these limits the electric load is practically independent on the outside temperature (we did not consider an "air-condition effect" in the Czech Republic). We also recognized the higher load sensitivity to the illumination during the spring and the autumn period. This sensitivity has the nonlinear character. On Fig. 3 we show the relationship between the daily electric load, the cloud cover variables and the season represented by a number of the week. This relationship is modelled by the neural network with two inputs. Because the complexity of our neural network is very low, the model is easily understandable and can be used as a component in our explanatory forecasting model.

The model was implemented in the power distribution company. The forecasting errors varies around 2 % which is very close to the errors of the measurement devices.

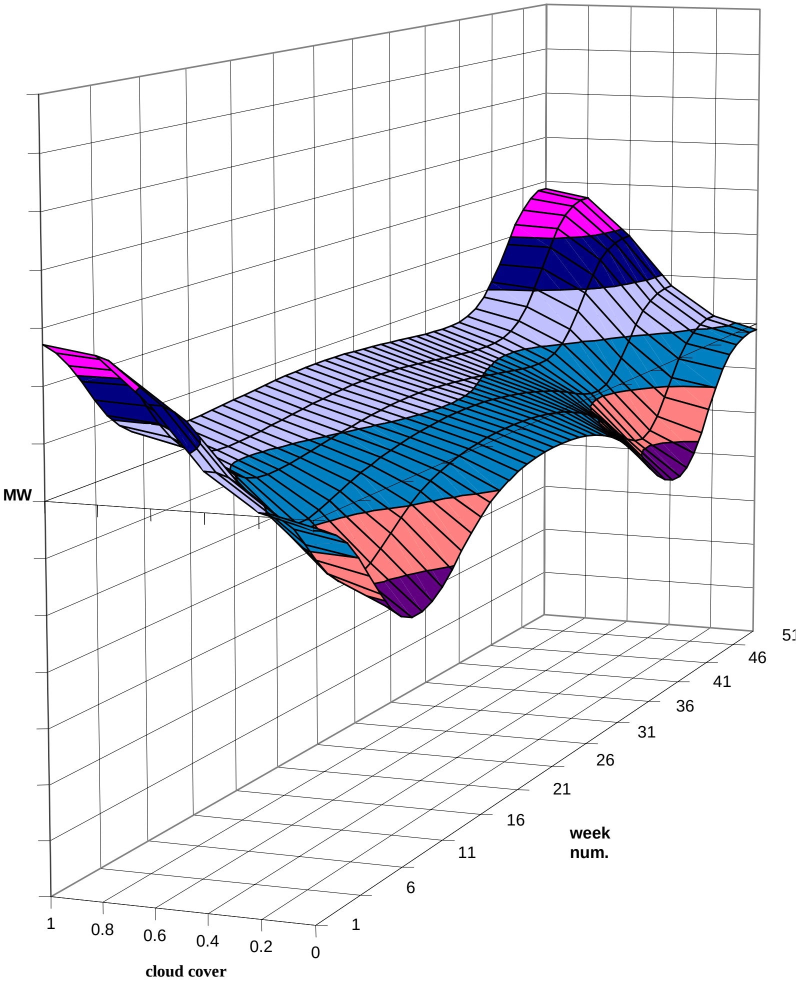


Fig. 3. Relationship between the daily electric load, the cloud cover variables (0 – bright sky, 1 – overcast) and the season (week number during a year) modelled by a neural network with two inputs

11 Conclusion

In this paper we summarize some basic steps of the forecasting methodology including identification and description of the problem, prediction quality specification, choosing a forecasting model, fitting method specification, fitting and validation data set specification, model architecture optimizing, testing, model implementation and using. We also mention some legal and economical aspect of forecasting as well some forecasting software, journals, conferences and internet addresses. Using two examples from banking and energy sector, we demonstrate different approaches in developing of the forecasting models.

We can conclude that regardless of the huge progress in forecasting approaches, mathematical complexity of the model, the statistical sophistication of the method, large databases and the power of computers, the computerized forecasting cannot substitute a human intuition. But, as support decision tools, these methods play a very valuable role in the information and knowledge processing area. Our forecast is that this will be true also in the future.

Acknowledgement

Author is grateful to the West-Bohemian Power Company Pilsen, Olsen & Associates Zürich for their data sets, Laboratory of Intelligent Systems (University of Economics) Prague, Laboratory of System Reliabiliy (Faculty of Transportation, Prague) for their fruitful collaboration and to the Grant Agency of the Ministry of Education, Youth and Sports (Grants VS96008) for its partial financial support.

References

1. Anděl, J.: Statistical Analysis of Time Series. SNTL, Prague (1976) (in Czech).
2. Armstrong, J. S.: Research Needs in Forecasting. International Journal of Forecasting **4** (1988) 449–465.
3. Armstrong, J. S., Schultz, R.: Principles Involving Marketing Policies: An Empirical Assessment. Marketing Letters **4** (1993) 253–265.
4. Armstrong, J.S.: Peer Review for Journals: Evidence on Quality Control, Fairness and Innovation. Science and Engineering Ethics **3** (1997) 63–84.
5. Armstrong J.S. (Ed.): Principles of Forecasting: A Handbook for Researchers and Practitioners. Norwell, MA: Kluwer Academic Publishers, in print. 313
6. Berelson, B., Steiner, G. A.: Human Behavior: An Inventory of Scientific Findings. New York: Harcourt, Brace & World (1964).
7. Böhm, V., Novák, M., Pelikán, E., Posseltová, V.: Neural Network Predicting System for Electric Load Data in West Bohemia. Proceedings of the Intelligent System Application to Power System, Montpellier, France (1994) 857–863.
8. Box, G. E. P., Jenkins, G. M.: Time Series Analysis: Forecasting and Control. Holden-Day, San Francisko, Revised Edition (1976).
9. Brown, R. G.: Statistical Forecasting for Inventory Control. New York: McGraw Hill (1959). 312

10. Bunn, D. W., Farmer E. D. (Eds.): Comparative Models for Electrical Load Forecasting. New York, John Willey (1985). 313
11. Casdagli, M.: Nonlinear Predictions of Chaotic Time Series. Physica D35 (1989) 335.
12. Chatfield, C.: Neural Networks: Forecasting Breakthrough or Passing Fad? International Journal of Forecasting **9** 1 (1993) 1–3.
13. Christ, C. F.: Simultaneous Equation Estimation: Any Verdict yet? Econometrica **28** (1960) 835–845.
14. Cipra, T.: Time Series Analysis with Application in Economy. SNTL, Prague (1986) (in Czech).
15. Cowles, A.: Can Stock Market Forecasters Forecast? Econometrica **1** (1933) 309–324.
16. Darbellay, G.: A Nonlinear Correlation Measure and its Application to Financial Time Series. Neural Network World **5** 4 (1995) 401–406. 315
17. De Groot, C., Würtz, D.: Analysis of Univariate Time Series with Connectionist Nets – A Case Study of Two Classical Examples. Neurocomputing **3** (1991) 177–192.
18. Dorn, H. F.: Pitfalls in Population Forecasts and Projections. Journal of the American Statistical Association **45** (1950) 311–334. 311
19. Ferber, R.: Are Correlations any Quide to Predictive Value. Applied Statistics **5** (1956) 113–122. 312
20. Fern, E. F.: The Use of Focus Groups for Idea Generation: The Effects of Group Size, Acquaintanceship, and Moderator on Response Quantity and Quality. Journal of Marketing Research **19** (1982) 1–13.
21. Fildes, R., Makridakis, S.: The Impact of Empirical Accuracy Studies on Time Series Analysis and Forecasting. International Statistical Review **63** (1995) 289–308.
22. Glasser, D.: A Reconsideration of some Parole Prediction Factors. American Sociological Review **19** (1954) 335–340.
23. Gordon, K.: Group Judgments in the Field of Lifted Weights. Journal of Experimental Psychology **7** (1924) 398–400. 311
24. Gorr, W. L., Nagin, D., Szczypula, J.: Comparative Study of Artificial Neural Network and Statistical Models for Predicting Student Grade Point Averages. International Journal of Forecasting **10** (1994) 17–34. 317
25. Griffin, A., Hauser J. R.: The Voice of the Consumer. Marketing Science **12** (1993) 1–17.
26. Jarvik M. E.: Probability Learning and Negative Recency Effect in the Serial Anticipation of Alternative Symbols. Journal of Experimental Psychology (1952) 291–297. 311
27. Koopmans, T. C.: Measurement without Theory. Review of Economics and Statistics **29** (1947) 161–172.
28. Laband, D. N.: Is There Value-Added from the Review Process in Economics? Preliminary evidence from authors. Quarterly Journal of Economics (1990) 341–352.
29. Lapedes, A. S., Farber, R. M.: Nonlinear Signal Processing Using Neural Networks: Prediction and System Modelling. Technical report LA-UR-87-2662, Los Alamos National Laboratory (1987).
30. MacGregor, D.: The Major Determinants in the Prediction of Social Events. Journal of Abnormal and Social Psychology **3** (1938) 179–204. 311
31. Makridakis, S., Wheelwright, S. C., Hyndman, R. J.: Forecasting: Methods and Applications. Third edition, John Willey & Sons (1998). 316

32. Meehl, P. E.: Clinical Versus Statistical Prediction: A Theoretical Analysis and a Review of Evidence. Minneapolis: University of Minnesota Press (1954).
33. Ogburn, W. F.: Studies in Prediction and the Distortion of Reality. Social Forces **13** (1934) 224–229. 311
34. Ošmera, P.: Optimization of Neural Networks by Genetic Algorithms. Neural Network World **5** 6 (1995) 965–976. 315
35. Payne S.: The Art of Asking Questions. Princeton University Press (1951).
36. Reiss, A.J.: The Accuracy, Efficiency and Validity of a Prediction Instrument. American Journal of Sociology **56** (1951) 552–561.
37. Paluš, M.: Detecting Nonlinearity in Multivariate Time Series. In: Physics Letters A 213 (1996) 138–147. 315
38. Pelikán, E., Vošvrda, M.: On Covariance Coefficients Estimates of Finite Order Moving Average Process. Kybernetika **17** 2 (1981) 169.
39. Pelikán, E.: Spectral Analysis of ARMA Processes by Prony's Method. Kybernetika, **20** 4 (1984) 322.
40. Pelikán, E., de Groot, C., Würtz, D.: Power Consumption in West Bohemia: Improved Forecasts with Decorrelating Connectionist Networks. Neural Network World **2** (1992) 701–712.
41. Pelikán, E., Beran, H., Novák, M.: Time Series Prediction by Artificial Neural Networks, in Frontier Decision Support Concepts – Help Desks, Learning, Fuzzy Diagnoses, Quality Evaluation, Prediction, Evolution, (Eds.) Plantamura V. L., Soucek B., Visaggio G., Sixth-Generation Computer Technology Series, John Wiley & Sons (1994), 221–239.
42. Pelikán, E., Matějka, P., Sláma, M., Vinkler, K. Interactive Forecasting of the Electric Load Using Kohonen Self-Organizing Feature Maps. Proc. of the World Congress of Neural Networks, San Diego, California, (1996) 443–446.
43. Pelikán, E., Sláma, M., Berka, P.: Rule-Based Forecasting: An Approach to Rules Exploring by Learning, Proc. of the "European Conference on Signal Analysis and Prediction" ECSAP-97, Prague, June 24-27 (1997) 195–198.
44. Pelikán, E., Berka, P.: Data Mining Methods in Prediction. Proc. of 6th International Workshop on Parallel Applications in Statistics and Economics PASE'97, Mariánské Lázně, November 9–12 (1997) 115–128.
45. Pelikán, E.: Neural Network Technology Applications in the Czech Republic: Energy, Transportation and Banking, (invited lecture). Dept. of Statistics and Operational Research, Faculty of Sciences, University of Lisbon, Portugal, Seminar on May, 29, 1998 full text published in Proc. of 1st Prediction workshop NOSTRADAMUS'98, Zlín, September 22–23 (1998) 37–43.
46. Ripley, B. D.: Statistical Aspects of Neural Networks. Proceedings SemStat, Chapman & Hall, London.
47. Sanders, H. T.: Convention Center Follies. The Public Interest **132** (1998) 58–72.
48. Sarbin, T. R.: A Contribution to the Study of Actuarial and Individual Methods of Prediction. American Journal of Sociology **48** (1943) 593–602.
49. Stroop, J. R.: Is the Judgment of the Group Better than that of the Average Member of the Group? Journal of Experimental Psychology **15** (1932) 550–560.
50. Stuckert, R. P.: A Configurational Approach to Prediction. Sociometry **21** (1932) 225–237.
51. Taylor, J. W., Bunn, D. W: Investigating Improvements in the Accuracy of Prediction Intervals for Combinations of Forecasts: A Simulation Study. International Journal of Forecasting **15** 3 (1999) 325–340.
52. Theil, H.: Applied Economic Forecasting. Amsterdam, Nord-Holland Publishing Co. (1966). 312

53. Toda, N., Murai, N., Usui, S.: A Measure of Nonlinearity in Time Series Using Neural Network Prediction Model. Artificial Neural Networks **2**, Aleksander, I., Taylor, J. (Eds.), Elsevier Science Publishers B.V. (1992) 1117–1120. 315
54. Tsay, R.S.: Nonlinearity Tests for Time Series. Biometrika **73** 2 (1986) 461–466. 315
55. Weigend, A. S., Huberman, B. A., Rumelhart, D. E.: Predicting the Future: A Connectionist Approach. International Journal of Neural Systems **1** (1990) 193–209. 317
56. Wold, H., Jureen, L.: Demand Analysis: A Study in Econometrics. New York: John Wiley (1953).

UPV-Curry: An Incremental Curry Interpreter*

M. Alpuente, S. Escobar, and S. Lucas

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, E-46022 Valencia, Spain.
{alpuente,sescobar,slucas}@dsic.upv.es

Abstract. Functional logic programming integrates the best features of modern functional and logic languages. The multi-paradigm declarative language Curry is an extension of Haskell which is intended to become a standard in the area. In this paper, we present UPV-Curry, an efficient and quite complete implementation of Curry based on a new, incremental definition of its basic evaluation mechanism. We compare UPV-Curry with already existing implementations of other Curry interpreters.

1 Introduction

Functional logic languages combine the best features of the most important declarative programming paradigms, namely functional and logic programming (see [9] for a survey). Throughout this decade, many practical proposals have been made to amalgamate functional and logic programming languages. However, these languages have not succeeded in becoming widely used by the functional or logic programming communities. The multi-paradigm language Curry [10,8] is an extension of Haskell [12] which is supported by an international initiative to make it a standard in the area. In order to facilitate and extend the use of Curry, it is essential to make efficient and practical implementations available. Program transformation techniques whose goal is to derive better semantically equivalent programs have recently been adapted to their use in Curry [1,4,5]. In this paper, we consider a different approach for improving the execution of Curry programs, which relies on an incremental definition of the Curry operational machinery. We present UPV-Curry, a novel implementation of Curry which provides an almost complete implementation of the language. In Curry, each evaluation step for an expression e is performed on a *needed* redex of e, i.e., a subexpression of e which is an instance of a left-hand side l of a (possibly conditional) program equation $l|c = r$ and which is really necessary for a complete evaluation of e. As is usual in functional logic languages, it is allowed to adequately instantiate variable occurrences of e in order to reduce this redex. The overall process is called *needed narrowing* [2]. To implement UPV-Curry, we have developed an *incremental* optimization of the basic operational machinery of Curry which makes the pattern matching between program equations and

* This work has been partially supported by CICYT TIC 98-0445-C03-01 and Acción Integrada hispano-alemana HA1997-0073.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 331–339, 1999.

goal expressions faster and is also able to *locally* resume the search for a new redex after performing each single evaluation step. We provide an experimental comparison of our implementation and other existing Curry interpreters.

2 Preliminaries

We assume familiarity with basic notions of term rewriting [6] and functional logic programming [9]. We consider a *signature* Σ partitioned into a set $\mathcal{C}$ of *constructors* and a set $\mathcal{F}$ of (defined) *functions* or *operations*. The set of *terms* and *constructor terms* with *variables* (e.g., x, y, z) from $\mathcal{X}$ are denoted by $\mathcal{T}(\Sigma, \mathcal{X})$ and $\mathcal{T}(\mathcal{C}, \mathcal{X})$, respectively. We write $\overline{o_n}$ for the *list of objects* $o_1, \ldots, o_n$. The set of variables occurring in a term t is denoted by $\mathcal{V}ar(t)$. A term is *linear* if it does not contain multiple occurrences of one variable. $root(t)$ denotes the symbol at the root of the term t. If $root(t) \in \mathcal{F}$ ($root(t) \in \mathcal{C}$), the term t is said to be *operation-rooted* (*constructor-rooted*). A *pattern* is a linear term of the form $f(\overline{d_n})$ where $f \in \mathcal{F}$ is n-ary and $d_1, \ldots, d_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$.

A *position* p in a term t is a sequence of natural numbers (Λ denotes the empty sequence, i.e., the root position). Positions are ordered by the *prefix* ordering: $p \leq q$, if $\exists p'.(p.p' = q)$. $t|_p$ denotes the *subterm* of t at position p, and $t[s]_p$ denotes the result of *replacing* $t|_p$ by the term s (see [6] for details). $\mathcal{P}os(t)$ ($\mathcal{FP}os(t)$) is the set of positions (of operation-rooted subterms) of t.

We denote by $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ the *substitution* σ with $\sigma(x_i) = t_i$ for $i = 1, \ldots, n$ (with $x_i \neq x_j$ if $i \neq j$), and $\sigma(x) = x$ for all other variables x. The identity substitution is denoted by *id*. Substitutions are extended to morphisms on terms by $\sigma(f(\overline{t_n})) = f(\overline{\sigma(t_n)})$ for every term $f(\overline{t_n})$. Given terms t, s, we write $t \leq s$ if $\exists \sigma. s = \sigma(t)$; a *unifier* of s and t is a substitution σ with $\sigma(s) = \sigma(t)$.

3 Curry

Curry is a functional logic programming language that combines the best ideas of declarative languages such as Haskell [12] and SML [15] (functional languages), Gödel [11] and λProlog [16] (logic languages), and Babel [13] and $\mathcal{TOY}$ [14] (functional logic languages). More specifically, Curry includes higher-order features, a type system, a module system, modern evaluation strategies, non-determinism, (encapsulated) search, partial data structures, existential variables, constraints, and declarative I/O.

We give a schematic description of the syntax of Curry. More details can be found in [8]. We first show a simple example of a Curry program.

Example 1. The following Curry program:

```
data Nat = Z | S Nat
plus :: Nat -> Nat
plus Z     x = x           plus (S x) y = S (plus x y)
```

contains the declaration of the *data type* `Nat`, an (optional) *type declaration* of the function `plus`, and the set of equations describing the function `plus`.

User defined data types are collections of *values* which are terms built from the data constructors which are associated to the data type being considered. For instance, `Z` and `S` are the data constructors for the data type `Nat`, and `Z`, `S Z`, and `S (S Z)` are examples of values of `Nat`. A Curry program mainly consists of a collection of data type declarations and function definitions given by (conditional) equations $f\ d_1\ \cdots\ d_n\ \{\texttt{|}c\}\ \texttt{=}\ r$ where f is the function symbol which the equation (partially) defines, $d_1, \ldots, d_n$ are constructor terms with variables, c is an (optional) condition, and r is the result expression. The condition c can either be a boolean expression (in the usual sense of functional languages) or a constraint expression (i.e., a conjunction of equational constraints $e_1\texttt{=:=}e_2$ where both e_1 and e_2 are expressions). An expression has the form:

$$\begin{array}{lll}
e ::= & x & \%\ \text{variable} \\
& \mid c\ e_1\ \cdots\ e_k & \%\ \text{application of } n\text{-ary constructor } c\ (0 \leq k \leq n) \\
& \mid f\ e_1\ \cdots\ e_k & \%\ \text{application of } n\text{-ary function } f\ (0 \leq k \leq n) \\
& \mid \texttt{if}\ b\ \texttt{then}\ e_1\ \texttt{else}\ e_2 & \%\ \text{conditional expression} \\
& \mid \backslash\ \mathit{pattern}\ \texttt{->}\ e & \%\ \text{a lambda abstraction} \\
& \mid e_1\ \mathit{op}\ e_2 & \%\ \text{an infix operator application}
\end{array}$$

3.1 Operational Semantics of Curry

The operational principle of Curry uses definitional trees. Given a program $\mathcal{R}$, a *definitional tree* $\mathcal{T}$ with pattern π (notation, $\mathit{pattern}(\mathcal{T}) = \pi$) is an expression of the form[1] [5]:

$\mathcal{T} = \mathit{rule}(\pi = r')$ where $\pi = r'$ is a variant of a program equation $l = r \in \mathcal{R}$.
$\mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_n)$ where $\pi|_o \in \mathcal{X}$, $c_1, \ldots, c_n$ are different constructors for $n > 0$, and each $\mathcal{T}_i$ is a definitional tree with pattern $\pi[c_i(x_1, \ldots, x_k)]_o$ where k is the arity of c_i and $x_1, \ldots, x_k$ are new variables.

A defined symbol f is called *inductively sequential* if there exists a definitional tree $\mathcal{T}$ with pattern $f(x_1, \ldots, x_k)$ whose rule nodes contain all (and only) the program equations defining f. In this case, we say that $\mathcal{T}$ is a definitional tree of f. A program $\mathcal{R}$ is *inductively sequential* if all its defined function symbols are inductively sequential. An inductively sequential program can be viewed as a set of definitional trees, each defining a function symbol.

Example 2. Given the following function definition:

```
first Z     y      = []
first (S n) (x:xs) = x : (first n xs)
```

The associated definitional tree is:

$$\begin{array}{l}
\mathit{branch}(\texttt{first x y}, 1, \\
\qquad \mathit{rule}(\texttt{first Z y = []}), \\
\qquad \mathit{branch}(\texttt{first (S n) y}, 2, \mathit{rule}(\texttt{first (S n) (m:ms) = m:(first n ms)})))
\end{array}$$

[1] Due to lack of space, we ignore the *or* nodes as well as the residuation of function calls used in Curry [8]. Of course, they have been considered in our implementation of the UPV-Curry interpreter.

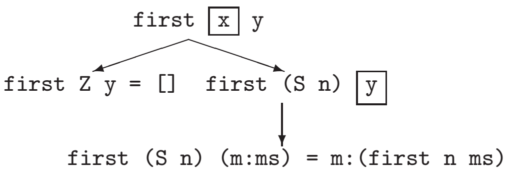


Fig. 1. Definitional tree for the function `first`

Figure 1 shows the graphical representation of this definitional tree.

A function λ from terms and definitional trees to sets of triples (p, R, σ) determines each evaluation step. Here, p is a position in t, R is the rule to be applied, and σ is a substitution which should be applied to t before reducing it. Given a term t and a definitional tree $\mathcal{T}$ such that $\mathit{pattern}(\mathcal{T}) \leq t$ [5]:

$$\lambda(t, \mathcal{T}) \ni \begin{cases} (\Lambda, l \to r, id) & \text{if } \mathcal{T} = \mathit{rule}(l = r); \\ (p, l \to r, \sigma \circ \tau) & \text{if } \mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k),\ t|_o = x \in \mathcal{X}, \\ & \quad \tau = \{x \mapsto c_i(\overline{x_n})\},\ \mathit{pattern}(\mathcal{T}_i) = \pi[c_i(\overline{x_n})]_o, \\ & \quad \text{and } (p, l \to r, \sigma) \in \lambda(\tau(t), \mathcal{T}_i); \\ (p, l \to r, \sigma) & \text{if } \mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k),\ t|_o = c_i(\overline{t_n}), \\ & \quad \mathit{pattern}(\mathcal{T}_i) = \pi[c_i(\overline{x_n})]_o, \\ & \quad \text{and } (p, l \to r, \sigma) \in \lambda(t, \mathcal{T}_i); \\ (o.p, l \to r, \sigma) & \text{if } \mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ & \quad t|_o = f(\overline{t_n}) \text{ for } f \in \mathcal{F}, \\ & \quad \mathcal{T}' \text{ is a definitional tree for } f, \\ & \quad \text{and } (p, l \to r, \sigma) \in \lambda(t|_o, \mathcal{T}') \end{cases}$$

If $(p, l{\to}r, \sigma) \in \lambda(t, \mathcal{T})$, then $t \leadsto_{(p,l\to r,\sigma)} s$ is a valid *narrowing* step, i.e., $l \leq \sigma(t|_p)$ and $s = \sigma(t[r]_p)$, which is called a *needed* narrowing step.

4 UPV-Curry

UPV-Curry is a novel interpreter of Curry which is publicly available from the URL: http://www.dsic.upv.es/users/elp/soft.html. UPV-Curry provides an almost complete implementation of Curry according to [8] (it lacks Curry modules and encapsulated search). The working environment of UPV-Curry provides the following facilities [7]:

- a small, self-contained Curry implementation, which constitutes a portable stand-alone SICStus Prolog application.
- a read-eval-point loop for displaying the solution for each expression which is entered as an input to the interpreter.
- simple browsing commands, which are able to obtain the type and the definitional tree of each function (see below).
- a debugging command, which permits tracing the evaluation of expressions.

4.1 Incremental Defininitional Trees

The implementation of UPV-Curry is based on a simpler representation of definitional trees. Given a program $\mathcal{R}$, an *incremental definitional tree* $\mathcal{I}$ with pattern π is an expression of the form:

$\mathcal{I} \;=\; \mathit{irule}(\pi = r')$ where $\pi = r'$ is a variant of a program equation $l = r \in \mathcal{R}$.
$\mathcal{I} \;=\; \mathit{ibranch}(o, (c_1, \mathcal{I}_1), \ldots, (c_n, \mathcal{I}_n))$ where $\pi|_o \in \mathcal{X}$, $c_1, \ldots, c_n$ are constructors for $n > 0$, and each $\mathcal{I}_i$ is an incremental definitional tree with pattern $\pi[c_i(x_1, \ldots, x_k)]_o$ where k is the arity of c_i, and $x_1, \ldots, x_k$ are new variables.

The pattern of an incremental definitional tree can be obtained as follows:

$$\mathit{pattern}(\mathcal{I}) = \begin{cases} \pi & \text{if } \mathcal{I} = \mathit{irule}(\pi = r); \\ \mathit{pattern}(\mathcal{I}_1)[x]_o & \text{if } \mathcal{I} = \mathit{ibranch}(o, (c_1, \mathcal{I}_1), \ldots, (c_n, \mathcal{I}_n)), \\ & \text{and } x \notin \mathcal{V}ar(\mathit{pattern}(\mathcal{I}_1)). \end{cases}$$

Standard and incremental definitional trees are related by a function ρ:

$$\rho(\mathcal{T}) = \begin{cases} \mathit{irule}(l = r) & \text{if } \mathcal{T} = \mathit{rule}(l = r); \\ \mathit{ibranch}(o, (c_1, \rho(\mathcal{T}_1)), \ldots, & \text{if } \mathcal{T} = \mathit{branch}(\pi, o, \mathcal{T}_1, \ldots, \mathcal{T}_k), \\ \qquad\qquad\quad (c_k, \rho(\mathcal{T}_k))) & \quad \text{and } \mathit{pattern}(\mathcal{T}_i) = \pi[c_i(\overline{x_n})]_o, \\ & \qquad 1 \le i \le k \end{cases}$$

Example 3. The definitional tree of Example 2 is represented as follows:

$\mathit{ibranch}(1, (\texttt{Z}, \mathit{irule}(\texttt{first Z y = []})),$
$\qquad\qquad (\texttt{S}, \mathit{ibranch}(2, (\texttt{:}, \mathit{irule}(\texttt{first (S n) (m:ms) = m:(first n ms)})))))$

Figure 2 shows the corresponding graphical representation.

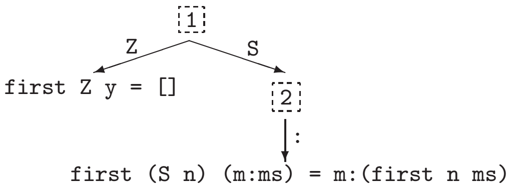


Fig. 2. Incremental definitional tree for the function `first`

In [3], we connect incremental definitional trees and different data structures which are used to guide needed reductions in functional programming.

4.2 Incremental Evaluation

Let $e_1 \rightsquigarrow_{(p_1,R_1,\sigma_1)} \cdots e_i \rightsquigarrow_{(p_i,R_i,\sigma_i)} e_{i+1} \cdots e_n$ be a needed narrowing sequence, where $(p_i, R_i, \sigma_i) \in \lambda(e_i, \mathcal{T}_i)$ and $\mathcal{T}_i$ is a definitional tree for $\mathit{root}(e_i)$, $i = 1, \ldots, n$. In the case when $e_{i+1}|_{p_i}$ is operation-rooted, we have that $p_{i+1} \geq p_i$, hence we take $(p_{i+1}, R_{i+1}, \sigma_{i+1}) = (p_i.p, R, \sigma)$, where $(p, R, \sigma) \in \lambda(e_{i+1}|_{p_i}, \mathcal{T})$ and $\mathcal{T}$ is a definitional tree for $\mathit{root}(e_{i+1}|_{p_i})$. If $e_{i+1}|_{p_i}$ is constructor-rooted or a variable, then it is necessary to resume the evaluation someplace above p_i, hence we go back to $e_{i+1}|_q$ (where $q < p_i$ is the position of the defined symbol immediately above p_i) by taking the ('most defined') definitional subtree $\mathcal{T}_i'$ of the definitional tree of $\mathit{root}(e_i|_q)$ which has been used in the computation of $\lambda(e_i, \mathcal{T}_i)$ immediately before obtaining $\mathcal{T}_i$. Thus, we let $(p_{i+1}, R_{i+1}, \sigma_{i+1}) = (q.p, R, \sigma)$, where $(p, R, \sigma) \in \lambda(e_{i+1}|_q, \mathcal{T}_i')$. This completes an incremental definition of λ.

We use a list $\mathcal{L}$ which contains the information needed to perform the needed narrowing steps incrementally; $\mathcal{L}$ is a list of pairs $(p, \mathcal{I})$ where p is a position and $\mathcal{I}$ is an incremental definitional tree. The *incremental needed narrowing strategy* is denoted by λ^ι. Given an operation-rooted term t and a list $\mathcal{L} = [(q, \mathcal{I}), \ldots]$ with $\mathit{pattern}(\mathcal{I}) \leq t|_q$, we compute $(p, R, \sigma, \mathcal{L}') \in \lambda^\iota(t, \mathcal{L})$, where (p, R, σ) describes a needed narrowing step and $\mathcal{L}'$ is a list containing the information which allows us to proceed with the subsequent evaluation steps incrementally. Formally,

$$\lambda^\iota(t, [(q,\mathcal{I})|\mathcal{L}]) \ni \begin{cases} (q, l \to r, id, \mathcal{L}') & \text{if } \mathcal{I} = \mathit{irule}(l = r) \text{ and} \\ & \quad \mathcal{L}' = \begin{cases} [(q,\mathcal{I}')|\mathcal{L}] & \text{if } \theta(r) \text{ is operation-rooted} \\ \mathcal{L} & \text{otherwise} \end{cases} \\ & \quad \text{where } t|_q = \theta(l) \text{ and } \mathcal{I}' \text{ is an incremental} \\ & \qquad\quad \text{definitional tree for } \mathit{root}(\theta(r)) \\ (p, l \to r, \sigma \circ \tau, \mathcal{L}') & \text{if } \mathcal{I} = \mathit{ibranch}(o, (c_1, \mathcal{I}_1), \ldots, (c_n, \mathcal{I}_n)), \\ & \quad t|_{q.o} = x \in \mathcal{X},\ \tau = \{x \mapsto c_i(\overline{x_k})\}, \\ & \quad \text{and } (p, l \to r, \sigma, \mathcal{L}') \in \lambda^\iota(\tau(t), [(q, \mathcal{I}_i)|\mathcal{L}]); \\ (p, l \to r, \sigma, \mathcal{L}') & \text{if } \mathcal{I} = \mathit{ibranch}(o, (c_1, \mathcal{I}_1), \ldots, (c_n, \mathcal{I}_n)), \\ & \quad t|_{q.o} = c_i(\overline{t_k}), \\ & \quad \text{and } (p, l \to r, \sigma, \mathcal{L}') \in \lambda^\iota(t, [(q, \mathcal{I}_i)|\mathcal{L}]); \\ (p, l \to r, \sigma, \mathcal{L}') & \text{if } \mathcal{I} = \mathit{ibranch}(o, (c_1, \mathcal{I}_1), \ldots, (c_n, \mathcal{I}_n)), \\ & \quad t|_{q.o} = f(\overline{t_k}) \text{ for } f \in \mathcal{F}, \\ & \quad \mathcal{I}' \text{ is an incremental definitional tree for } f, \\ & \quad \text{and } (p, l \to r, \sigma, \mathcal{L}') \in \lambda^\iota(t, [(q.o, \mathcal{I}'), (q, \mathcal{I})|\mathcal{L}]) \end{cases}$$

The needed narrowing steps performed by λ^ι and the computation steps carried out by the original needed narrowing strategy λ are equivalent.

Theorem 1. *Let $\mathcal{R}$ be an inductively sequential program, $e_1 \rightsquigarrow e_2 \rightsquigarrow \cdots \rightsquigarrow e_n$ be a needed narrowing derivation where $\mathcal{T}_i$ is a definitional tree such that $\mathit{pattern}(\mathcal{T}_i) \leq e_i$ for all $1 \leq i \leq n$, and $\mathcal{L}_0 = [(\Lambda, \rho(\mathcal{T}_1))]$. Then, for all $1 \leq i \leq n$, $(p_i, R_i, \sigma_i) \in \lambda(e_i, \mathcal{T}_i)$ if and only if $(p_i, R_i, \sigma_i, \mathcal{L}_i) \in \lambda^\iota(e_i, \mathcal{L}_{i-1})$.*

Table 1. Runtime goals and comparison with other Curry interpreters (in ms.)

Benchmark	Goal	UPV-Curry	TasteCurry	PACS-TasteCurry
iter	iter 100 sub1 100	323	1174	909
ackermann	ackermann 20	640	2505	1203
mergesort	sort (intMerge) [3,1,2] xs	313	870	442
quicksort	qsort $[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]$	2476	1829	1511
fibonacci	fibonacci 10	576	2262	1106
horseman	horseman x y 8 20	2655	3542	1658
last	last [1]$_{100}$	990	58736	4256

5 Experimental Results

In [3], we provide experimental evidence of the fact that all the modifications described in the previous sections are actually (and independently) effective by comparing a preliminary, non-optimized version of UPV-Curry and different versions of the interpreter including incremental definitional trees and incremental evaluation. In this section, we compare UPV-Curry and other Curry interpreters. Table 1 provides the runtimes of the benchmarks for UPV-Curry, TasteCurry, and PACS-TasteCurry interpreters[2]. Times were measured on a SUN SparcStation, running under UNIX System V Release 4.0. They are expressed in milliseconds and are the average of 10 executions. Most of the benchmarks used for the analysis (namely, quicksort, last, horseman, and mergesort) are standard Curry test programs[3]. The benchmarks ackermann, fibonacci, and iter are given in Table 2. The functions ackermann, fibonacci, last, quicksort and mergesort, are well-known; horseman computes the number of men and horses that have a certain number of heads and feet; finally, iter produces a sequence of n nested calls to a given function. Natural numbers are implemented using Z/S-terms, and lists are shown in goals using a subindex which represents its size. The figures in Table 1 show that our UPV-Curry implementation performs very well in comparison to the other Curry interpreters. In overall, it takes about 80 % the time needed by PACS-TasteCurry and about 38 % the time needed by TasteCurry to evaluate the queries. These results seem to substantiate the advantages of using the proposed incremental techniques. A more detailed comparison among Curry implementations – including also more benchmark examples – is underway.

6 Conclusions

We have presented UPV-Curry, a novel and effective implementation of Curry which provides an almost complete implementation of the language. In order to

[2] TasteCurry and PACS-TasteCurry are available from http://www-i2.informatik.rwth-aachen.de/~hanus/curry/.

[3] Look at http://www-i2.informatik.rwth-aachen.de/~hanus/curry/examples/ for a collection of examples containing these and other available Curry test programs.

Table 2. Benchmark code

ackermann	

`ackermann n = ack (S (S Z)) n`	`ack Z n = S n`
	`ack (S m) Z = ack m (S Z)`
	`ack (S m) (S n) = ack m (ack (S m) n)`

fibonacci	

`sum Z y = y`
`sum (S x) y = S (sum x y)`
	`fib Z = S Z`
	`fib (S (S x)) = sum (fib (S x)) (fib x)`
	`fib (S x) = S x`

iter	

`iter Z f x = x`
`iter (S y) f x = f (iter y f x)`
	`sub1 (S Z) = Z`
	`sub1 (S (S Z)) = S Z`
	`sub1 (S (S (S Z))) = S (S Z)`
	`sub1 (S (S (S (S x)))) = S (S (S x))`

improve performance, we have implemented (and proved correct) an *incremental* (re)definition of the basic evaluation strategy of Curry which is able to *locally* resume the search for a new redex after performing a revious incremental narrowing step, and also reduces the number of term positions to be considered in a needed narrowing derivation. Finally, we have provided the (first) experimental comparison of different implementations of existing Curry interpreters.

References

1. E. Albert, M. Alpuente, M. Hanus, and G. Vidal. A Partial Evaluation Framework for Curry Programs. In *Proc. of LPAR '99*. 331
2. S. Antoy, R. Echahed, and M. Hanus. A Needed Narrowing Strategy. In *Proc. of POPL '94*, pages 268–279, 1994. 331
3. M. Alpuente, S. Escobar, and S. Lucas. Incremental needed narrowing. In *Proc. of the International PLI '99 Workshop IDL '99*. 335, 337
4. M. Alpuente, M. Falaschi, G. Moreno, and G. Vidal. A Transformation System for Lazy Functional Logic Programs In *Proc. of FLOPS '99*. 331
5. M. Alpuente, M. Hanus, S. Lucas, and G. Vidal. Specialization of Inductively Sequential Functional Logic Programs. In *Proc. of ICFP '99*. 331, 333, 334
6. F. Baader and T. Nipkow. Term Rewriting and All That. Cambridge University Press, 1998. 332, 332
7. S. Escobar, M. Alpuente, and S. Lucas. UPV-Curry User's Manual. TR DSIC-II/38/98. http://www.dsic.upv.es/users/elp/papers.html. 334
8. M. Hanus, S. Antoy, H. Kuchen, F. J. López-Fraguas, and F. Steiner. Curry An Integrated Functional Logic Language (version 0.5). Available at http://www-i2.informatik.rwth-aachen.de/ hanus/curry, Jan 1999. 331, 332, 333, 334
9. M. Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19&20:583–628, 1994. 331, 332
10. M. Hanus, H. Kuchen, and J.J. Moreno-Navarro. Curry: A truly functional logic language. In *Proc. ILPS'95 Workshop on Visions for the Future of Logic Programming*, pages 95–107, 1995. 331

11. P. Hill and J. W. Lloyd. *The Gödel Programming Language.* The MIT Press, Cambridge, MA, 1994. 332
12. P. Hudak, S. Peyton Jones, and P. Wadler. Report on the Functional Programming Language Haskell: a non-strict, purely functional language. *Sigplan Notices*, 27(5), 1992. 331, 332
13. H. Kuchen and J. Anastasiadis. Higher Order Babel: Language and implementation. In *Proc. of ELP '96*, LNAI 1050:193–207, 1996. 332
14. F.J. López Fraguas and J. Sánchez Hernández. $\mathcal{TOY}$: A multiparadigm Declarative System. In *Proc. of RTA '99*, LNCS 1631, 1999. 332
15. R. Milner, M. Tofte, and R. Harper. *The Definition of ML.* The MIT Press, 1990. 332
16. G. Nadathur and D. Miller. An Overview of λProlog. In *Proc. of ICLP'88*, pages 810–827. The MIT Press, Cambridge, MA, 1988. 332

Quantum Finite Multitape Automata

Andris Ambainis[1]*, Richard Bonner[2], Rūsiņš Freivalds[3]**, Marats Golovkins[3]***, and Marek Karpinski[4]†

[1] Computer Science Division, University of California, Berkeley, CA 94720-2320
ambainis@cs.berkeley.edu

[2] Department of Mathematics and Physics, Mälardalens University
richard.bonner@mdh.se

[3] Institute of Mathematics and Computer Science, University of Latvia
Raiņa bulv. 29, Riga, Latvia
rusins@cclu.lv
marats@cclu.lv

[4] Department of Computer Science, University of Bonn
53117, Bonn, Germany
marek@cs.bonn.edu

Abstract. Quantum finite automata were introduced by C. Moore, J. P. Crutchfield [4], and by A. Kondacs and J. Watrous [3]. This notion is not a generalization of the deterministic finite automata. Moreover, in [3] it was proved that not all regular languages can be recognized by quantum finite automata. A. Ambainis and R. Freivalds [1] proved that for some languages quantum finite automata may be exponentially more concise rather than both deterministic and probabilistic finite automata. In this paper we introduce the notion of quantum finite multitape automata and prove that there is a language recognized by a quantum finite automaton but not by deterministic or probabilistic finite automata. This is the first result on a problem which can be solved by a quantum computer but not by a deterministic or probabilistic computer. Additionally we discover unexpected probabilistic automata recognizing complicated languages.

1 Introduction

The basic model, i.e., quantum finite automata (QFA), were introduced twice. First this was done by C. Moore and J. P. Crutchfield [4]. Later in a different and non-equivalent way these automata were introduced by A. Kondacs and J. Watrous [3].

* Supported by Berkeley Fellowship for Graduate Studies.

** Research supported by Grant No. 96.0282 from the Latvian Council of Science

*** Research supported by Grant No. 96.0282 from the Latvian Council of Science

† Research partially supported by the International Computer Science Institute, Berkeley, California, by the DFG grant KA 673/4-1, and by the ESPRIT BR Grants 7079 and ECUS030

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 340–348, 1999.

The first definition just mimics the definition of 1-way finite probabilistic only substituting *stochastic* matrices by *unitary* ones. To define quantum finite multitape automata we generalize a more elaborated definition [3].

We are using these notations in the following definition:
z^* is the complex conjugate of a complex number z.
$M =_{def} \{1, 2, \ldots, m\}$.
The k-th component of an arbitrary vector s will be defined as s^k.
We shall understand by I an arbitrary element from the set $\mathcal{P}(M) \setminus \{\emptyset\}$.
$R_I =_{def} A_1 \times A_2 \times \cdots \times A_m$, where $A_i = \begin{cases} \{\downarrow, \rightarrow\}, & \text{if } i \notin I \\ \{\text{"}nothing\text{"}\}, & \text{if } i \in I. \end{cases}$
$T_I =_{def} B_1 \times B_2 \times \cdots \times B_m$, where $B_i = \begin{cases} \{\downarrow, \rightarrow\}, & \text{if } i \in I \\ \{\text{"}nothing\text{"}\}, & \text{if } i \notin I. \end{cases}$
The function $R_i \times T_i \xrightarrow{d_I} \{\downarrow, \rightarrow\}^m$ is defined as follows:
$d_I(r,t) =_{def} (d_I^1(r,t), d_I^2(r,t), \ldots, d_I^m(r,t))$, where $d_I^i(r,t) = \begin{cases} r^i, & \text{if } i \notin I \\ t^i, & \text{if } i \in I. \end{cases}$

Definition 1. *A quantum finite multitape automaton (QFMA) $A = (Q; \Sigma; \delta; q_0; Q_a; Q_r)$ is specified by the finite input alphabet Σ, the finite set of states Q, the initial state $q_0 \in Q$, the sets $Q_a \subset Q$, $Q_r \subset Q$ of accepting and rejecting states, respectively, with $Q_a \cap Q_r = \emptyset$, and the transition function*

$$\delta \colon Q \times \Gamma^m \times Q \times \{\downarrow, \rightarrow\}^m \longrightarrow \mathbb{C}_{[0,1]},$$

where m is the number of input tapes, $\Gamma = \Sigma \cup \{\#, \$\}$ is the tape alphabet of A and #,\$ are end-markers not in Σ, which satisfies the following conditions (of well-formedness):

1. *Local probability condition.*

$$\forall (q_1, \sigma) \in Q \times \Gamma^m \colon \sum_{(q,d) \in Q \times \{\downarrow, \rightarrow\}^m} |\delta(q_1, \sigma, q, d)|^2 = 1.$$

2. *Orthogonality of column vectors condition.*

$$\forall q_1, q_2 \in Q, q_1 \neq q_2, \forall \sigma \in \Gamma^m \colon \sum_{(q,d) \in Q \times \{\downarrow, \rightarrow\}^m} \delta^*(q_1, \sigma, q, d) \delta(q_2, \sigma, q, d) = 0.$$

3. *Separability condition.*

$$\forall I \in \mathcal{P}(M) \setminus \{\emptyset\} \ \forall q_1, q_2 \in Q$$
$$\forall \sigma_1, \sigma_2 \in \Gamma^m, \textit{where } \forall i \notin I \ \sigma_1^i = \sigma_2^i$$
$$\forall t_1, t_2 \in T_I, \ \textit{where } \forall j \in I \ t_1^j \neq t_2^j$$
$$\sum_{(q,r) \in Q \times R_I} \delta^*(q_1, \sigma_1, q, d_I(r, t_1)) \delta(q_2, \sigma_2, q, d_I(r, t_2)) = 0.$$

States from $Q_a \cup Q_r$ are called halting states and states from $Q_{non} = Q \setminus (Q_a \cup Q_r)$ are called non-halting states.

To process an input word vector $x \in (\Sigma^*)^m$ by A it is assumed that the input is written on every tape k with the end-markers in the form $w_x^k = \#x^k\$$ and that every such a tape, of length $|x^k|+2$, is circular, i. e., the symbol to the right of $\$$ is $\#$.

For the fixed input word vector x we can define $n \in \mathbb{N}^m$ to be an integer vector which determines the length of input word on every tape. So for every n we can define C_n to be the set of all possible configurations of A where $|x^i| = n^i$. $|C_n| = |Q| \prod_{i=1}^{m} (n^i + 2)$. Every such a configuration is uniquely determined by a pair $|q, s\rangle$, where $q \in Q$ and $0 \leq s^i \leq |x^i| + 1$ specifies the position of head on the i-th tape.

Every computation of A on an input x, $|x^i| = n^i$, is specified by a unitary evolution in the Hilbert space $H_{A,n} = l_2(C_n)$. Each configuration $c \in C_n$ corresponds to the basis vector in $H_{A,n}$. Therefore a global state of A in the space $H_{A,n}$ has a form $\sum_{c \in C_n} \alpha_c |c\rangle$, where $\sum_{c \in C_n} |\alpha_c|^2 = 1$. If the input word vector is x and the automaton A is in its global state $|\psi\rangle = \sum_{c \in C_n} \alpha_c |c\rangle$, then its further step is equivalent to the application of a linear operator U_x^δ over Hilbert space $l_2(C_n)$.

Definition 2. *The linear operator U_x^δ is defined as follows:*

$$U_x^\delta |\psi\rangle = \sum_{c \in C_n} \alpha_c U_x^\delta |c\rangle .$$

If a configuration $c = |q', s\rangle$, then

$$U_x^\delta |c\rangle = \sum_{(q,d) \in Q \times \{\downarrow, \rightarrow\}^m} \delta(q', \sigma(s), q, d) \mid q, \tau(s, d)\rangle ,$$

where $\sigma(s) = (\sigma^1(s), \ldots, \sigma^m(s))$, $\sigma^i(s)$ specifies the s^i-th symbol on the i-th tape, and $\tau(s,d) = (\tau^1(s,d), \ldots, \tau^m(s,d))$,

$$\tau^i(s,d) = \begin{cases} (s^i + 1) \bmod (n^i + 2), & if\ d^i = `\rightarrow' \\ s^i, \ if\ d^i = `\downarrow' . \end{cases}$$

Lemma 1. *The well-formedness conditions are satisfied iff for any input x the mapping U_x^δ is unitary.*

Language recognition for QFMA is defined as follows. For each input x with the corresponding vector n, $n^i = |x^i|$, and a QFMA $A = (Q; \Sigma; \delta; q_0; Q_a; Q_r)$ we define $C_n^a = \{(q,s) \mid (q,s) \in C_n, q \in Q_a\}$, $C_n^r = \{(q,s) \mid (q,s) \in C_n, q \in Q_r\}$, $C_n^{non} = C_n \setminus (C_n^a \cup C_n^r)$. E_a, E_r, E_{non} are the subspaces of $l_2(C_n)$ spanned by C_n^a, C_n^r, C_n^{non} respectively. We use the observable $\mathcal{O}$ that corresponds to the orthogonal decomposition $l_2(C_n) = E_a \oplus E_r \oplus E_{non}$. The outcome of each observation is either "accept" or "reject" or "non-halting".

The language recognition is now defined as follows: For an $x \in (\Sigma^*)^m$ we consider as the input ω_x, $\omega_x^k = \#x^k\$$, and assume that the computation starts with A being in the configuration $|q_0, \{0\}^m\rangle$. Each computation step consists of two parts. At first the linear operator $U_{\omega_x}^\delta$ is applied to the current global state and then the resulting superposition, i.e., global state, is observed using the observable $\mathcal{O}$ as defined above. If the global state before the observation is $\sum\limits_{c \in C_n} \alpha_c |c\rangle$, then the probability that the subspace E_i, $i \in \{a, r, non\}$, will be chosen is $\sum\limits_{c \in C_n^i} |\alpha_c|^2$. The computation continues until the result of an observation is "accept" or "reject".

Definition 3. *A QFMA $A = (Q; \Sigma; \delta; q_0; Q_a; Q_r)$ is simple if for each $\sigma \in \Gamma^m$ there is a linear unitary operator V_σ over the inner-product space $l_2(Q)$ and a function $D\colon Q \longrightarrow \{\downarrow, \rightarrow\}^m$, such that*

$$\forall q_1 \in Q \;\forall \sigma \in \Gamma^m \quad \delta(q_1, \sigma, q, d) = \begin{cases} \langle q|V_\sigma|q_1\rangle, & \text{if } D(q) = d \\ 0, & \text{otherwise.} \end{cases}$$

Lemma 2. *If the automaton A is simple, then conditions of well-formedness are satisfied iff for every σ V_σ is unitary.*

We shall deal only with simple multitape automata further in the paper.

2 Quantum vs. Probabilistic Automata

Definition 4. *We shall say that an automaton is deterministic reversible finite multitape automaton (RFMA), if it is a simple QFMA with $\delta(q_1, \sigma, q, d) \in \{0, 1\}$.*

Definition 5. *We say that a language L is $[m, n]$-deterministically recognizable if there are n deterministic automata A_1, A_2, A_n such that:*

a) if the input is in the language L, then all n automata $A_1, \ldots, A_n$ accept the input;
b) if the input is not in the language L, then at most m of the automata $A_1, \ldots, A_n$ accept the input.

Definition 6. *We say that a language L is $[m, n]$-reversibly recognizable if there are n deterministic reversible automata A_1, A_2, A_n such that:*

a) if the input is in the language L, then all n automata $A_1, \ldots, A_n$ accept the input;
b) if the input is not in the language L, then at most m of the automata $A_1, \ldots, A_n$ accept the input.

Lemma 3. *If a language L is $[1,n]$-deterministically recognizable by 2-tape finite automata, then L is recognizable by a probabilistic 2-tape finite automaton with probability $\frac{n}{n+1}$.*

Proof. The probabilistic automaton starts by choosing a random integer $1 \leq r \leq (n+1)$. After that, if $r \leq n$, then the automaton goes on simulating the deterministic automaton A_r, and, if $r = n+1$, then the automaton rejects the input. The inputs in L are accepted with probability $\frac{n}{n+1}$, and the inputs not in the language are rejected with a probability no less than $\frac{n}{n+1}$. □

Lemma 4. *If a language L is $[1,n]$-reversibly recognizable by 2-tape finite automata, then L is recognizable by a quantum 2-tape finite automaton with probability $\frac{n}{n+1}$.*

Proof. In essence the algorithm is the same as in Lemma 3. The automaton starts by taking $n+1$ different actions with amplitudes $\frac{1}{\sqrt{n+1}}$. (It is possible to construct a unitary matrix to make such a choice feasible.) After that the automaton simultaneously goes on simulating all the deterministic reversible automata A_r, $1 \leq r \leq (n+1)$, where the automaton A_{n+1} rejects an input. The simulation of each deterministic reversible automaton uses its own accepting and rejecting states. (Hence the probabilities are totaled, not the amplitudes.) □

First, we discuss the following 2-tape language

$$L_1 = \{(x_1 \nabla x_2, y) \mid x_1 = x_2 = y\},$$

where the words x_1, x_2, y are unary.

Lemma 5. *(Proved by R. Freivalds [2].) For arbitrary natural n, the language L_1 is $[1,n]$-deterministically recognizable.*

Lemma 6. *For arbitrary natural n, the language L_1 is $[1,n]$-reversibly recognizable.*

Proof. By Lemma 5, the language L_1 is $[1,n]$-deterministically recognizable. However it is easy enough to make the construction of the automata $A_1, \ldots, A_n$ in the following manner:

a) every automaton is reversible;
b) if a word pair is in the language L_1, then every automaton consumes the same number of steps to accept the word pair.

The last requirement will be essential further in the paper. If at least the first requirement is met, then the language is $[1,n]$-reversibly recognizable. □

Theorem 1. *The language L_1 can be recognized with arbitrary probability $1-\epsilon$ by a probabilistic 2-tape finite automaton but this language cannot be recognized by a deterministic 2-tape finite automaton.*

Theorem 2. *The language L_1 can be recognized with arbitrary probability $1-\epsilon$ by a quantum 2-tape finite automaton.*

Proof. By Lemmas 4 and 6. □

In an attempt to construct a 2-tape language recognizable by a quantum 2-tape finite automaton but not by probabilistic 2-tape finite automata we consider a similar language

$$L_2 = \{(x_1 \nabla x_2 \nabla x_3, y) \mid \text{ there are exactly 2 values of } x_1, x_2, x_3 \text{ such that they equal } y\},$$

where the words x_1, x_2, x_3, y are unary.

Theorem 3. *A quantum automaton exists which recognises the language L_2 with a probability $\frac{9}{16} - \epsilon$ for arbitrary positive ϵ.*

Proof. This automaton takes the following actions with the following amplitudes:

a) $\frac{\sqrt{3}}{4} \cdot 1$ – compares $x_1 = x_2 = y$,
b) $\frac{\sqrt{3}}{4} \cdot (\cos \frac{2\pi}{3} + i \sin \frac{2\pi}{3})$ – compares $x_2 = x_3 = y$,
c) $\frac{\sqrt{3}}{4} \cdot (\cos \frac{4\pi}{3} + i \sin \frac{4\pi}{3})$ – compares $x_1 = x_3 = y$,
d) $\frac{\sqrt{7}}{4}$ – says “accept”.

By Theorem 2 comparison in actions a), b), c) can be accomplished. By construction in Lemma 4 the comparison in each action a), b), c) is implemented by starting $n+1$ different branches. Therefore in any action i), $i \in \{a, b, c\}$, if a comparison is successful, the automaton will come respectively into non-halting states $q_{a,1}, \dots, q_{a,n}$, $q_{b,1}, \dots, q_{b,n}$, $q_{c,1}, \dots, q_{c,n}$, reaching the symbol pair $(\$, \$)$ on the tapes. The transition $(\$, \$)$ for every $k = 1, \dots, n$ is as follows:

	$q_{a,k}$	$q_{b,k}$	$q_{c,k}$
$q_{a1,k}$	$\frac{1}{\sqrt{3}}$	$\frac{1}{\sqrt{3}}$	$\frac{1}{\sqrt{3}}$
$q_{r,k}$	$\frac{1}{\sqrt{3}}$	$\frac{1}{\sqrt{3}}(\cos \frac{4\pi}{3} + i \sin \frac{4\pi}{3})$	$\frac{1}{\sqrt{3}}(\cos \frac{2\pi}{3} + i \sin \frac{2\pi}{3})$
$q_{a2,k}$	$\frac{1}{\sqrt{3}}$	$\frac{1}{\sqrt{3}}(\cos \frac{2\pi}{3} + i \sin \frac{2\pi}{3})$	$\frac{1}{\sqrt{3}}(\cos \frac{4\pi}{3} + i \sin \frac{4\pi}{3})$

Here $q_{a1,k}, q_{a2,k}$ are accepting states and $q_{r,k}$ are rejecting states. If y equals all 3 words x_1, x_2, x_3, then it is possible to ensure that it takes the same time to reach the end-marking symbol pair in every action on every branch. Therefore the input is accepted with probability $\frac{7}{16} + \epsilon$ (since the amplitudes of the actions a), b), c) total to 0). If y equals 2 out of 3 words x_1, x_2, x_3, then the input is accepted with probability $\frac{9}{16} - \epsilon$. If y equals at most one of the words x_1, x_2, x_3, then the input is accepted with probability $\frac{7}{16} + \epsilon$ (only if the action d) is taken). □

Unfortunately, the following theorem holds.

Theorem 4. *A probabilistic automaton exists which recognizes the language L_2 with a probability $\frac{21}{40}$.*

Proof. The probabilistic automaton with probability $\frac{1}{2}$ takes an action A or B:

A) Choose a random j and compare $x_j = y$. If **yes**, accept with probability $\frac{19}{20}$. If **no**, accept with probability $\frac{1}{20}$.
B) Choose a random pair j, k and compare $x_j = x_k = y$. If **yes**, reject. If **no**, accept with probability $\frac{12}{20}$.

If y equals all 3 words x_1, x_2, x_3 and the action A is taken, then the input is accepted with relative probability $\frac{19}{20}$. If y equals all 3 words x_1, x_2, x_3 and the action A is taken, then the input is accepted with relative probability 0. This gives the acceptance probability in the case if y equals all 3 words x_1, x_2, x_3, to be $\frac{19}{40}$ and the probability of the correct result "no" to be $\frac{21}{40}$.

If y equals 2 words out of x_1, x_2, x_3 and the action A is taken, then the input is accepted with relative probability $\frac{13}{20}$. If y equals 2 words out of x_1, x_2, x_3 and the action B is taken, then the input is accepted with relative probability $\frac{8}{20}$. This gives the acceptance probability in the case if y equals 2 words out of x_1, x_2, x_3, to be $\frac{21}{40}$.

If y equals only 1 word out of x_1, x_2, x_3 and the action A is taken, then the input is accepted with relative probability $\frac{7}{20}$. If y equals only 1 word out of x_1, x_2, x_3 and the action B is taken, then the input is accepted with relative probability $\frac{12}{20}$. This gives the acceptance probability in the case if y equals only 1 word out of x_1, x_2, x_3, to be $\frac{19}{40}$ and the probability of the correct result "no" to be $\frac{21}{40}$.

If y equals no word of x_1, x_2, x_3 and the action A is taken, then the input is accepted with relative probability $\frac{1}{20}$. If y equals no word of x_1, x_2, x_3 and the action B is taken, then the input is accepted with relative probability $\frac{12}{20}$. This gives the acceptance probability in the case if y equals no word of x_1, x_2, x_3, to be $\frac{13}{40}$ and the probability of the correct result "no" to be $\frac{27}{40}$. □

Now we consider a modification of the language L_2 which might be more difficult for a probabilistic recognition:

$$\begin{aligned} L_3 = \{(x_1 \nabla x_2 \nabla x_3, y_1 \nabla y_2) \mid & \text{ there is exactly one value } k \\ & \text{such that there are exactly two values } j \text{ such that } x_j = y_k\} \end{aligned}$$

Theorem 5. *A quantum finite 2-tape automaton exists which recognizes the language L_3 with a probability $\frac{36}{67} - \epsilon$ for arbitrary positive ϵ.*

However this language also can be recognized by a probabilistic 2-tape finite automaton.

Theorem 6. *A probabilistic finite 2-tape automaton exists which recognizes the language L_3 with a probability $\frac{13}{25} - \epsilon$ for arbitrary positive ϵ.*

Proof. The probabilistic automaton with probability $\frac{6}{25}$ takes action A or B or C or with probability $\frac{7}{25}$ takes action D:

A) Choose a random k and two values of j. Then compare $x_j = y_k$. If **yes**, accept. If **no**, reject.

B) Chose a random k and compare $x_1 = x_2 = x_3 = y_k$. If **yes**, reject. If **no**, accept.
C) Choose two values j and m. Then compare $x_j = x_m = y_1 = y_2$. If **yes**, reject. If **no**, accept.
D) Says "reject".

Notice that the actions A, B, C are probabilistic, and they can be performed only with probability $1 - \epsilon$ (actions A and B are described in the proof of Theorem 1 and action C is similar).
The acceptance probabilities equal:

	A	B	C	total
no y_k equals 2 or 3 x_j	0	1	1	$\frac{12}{25}$
one y_k equals 2 x_j	$\frac{1}{6}$	1	1	$\frac{13}{25}$
one y_k equals 3 x_j	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{12}{25}$
two y_k equal 2 x_j	$\frac{1}{3}$	1	$\frac{2}{3}$	$\frac{12}{25}$
all y_k equal all x_j	1	0	0	$\frac{6}{25}$

□

Finally we consider a modification of the languages above which recognition indeed is impossible by probabilistic automata:

$$L_4 = \{(x_1 \nabla x_2, y) \mid \text{there is exactly one value } j \text{ such that } x_j = y\}$$

where the words x_1, x_2, y are binary.

Theorem 7. *A quantum finite 2-tape automaton exists which recognizes the language L_4 with a probability $\frac{4}{7}$.*

Proof. The automaton has two accepting q_{a1}, q_{a2} and three rejecting states q_{r1}, q_{r2}, q_{r3} and starts the following actions by reading the pair $(\#, \#)$ with the following amplitudes:

a) with an amplitude $\sqrt{\frac{2}{7}}$ compares x_1 to y,
b) with an amplitude $-\sqrt{\frac{2}{7}}$ compares x_2 to y,
c) with an amplitude $\sqrt{\frac{3}{7}}$ immediately goes to the state q_{a1}.

Actions a) and b) use different non-halting states to process the word pair. All these actions the automaton processes simultaneously. In actions a) and b), if **no** (not equal), it goes accordingly to the states q_{r1} or q_{r2}, if **yes**, then reaches correspondent non-halting states q_α or q_β, while the symbol pair on the tapes is $(\$, \$)$. The transition for $(\$, \$)$ and states $q_\alpha, q_\beta, q_{a2}, q_{r3}$ is as follows:

	q_α	q_β
q_{a2}	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$
q_{r3}	$\frac{1}{\sqrt{2}}$	$-\frac{1}{\sqrt{2}}$

If all the words are equal, it is possible to ensure that it takes the same time to reach the end-markers on both tapes, therefore the automaton reaches the superposition $\sqrt{\frac{2}{7}}|q_\alpha, s, t\rangle - \sqrt{\frac{2}{7}}|q_\beta, s, t\rangle$, where s and t specify the place of \$ on each tape, and the input is accepted with probability $\frac{3}{7}$. (Since the amplitudes of the actions a) and b) equal to 0.) If one of the words x_i equals y, then the input is accepted with probability $\frac{4}{7}$. If none of the words x_i equals y, then the input is accepted with probability $\frac{3}{7}$. □

References

1. Andris Ambainis and Rūsiņš Freivalds. 1-way quantum finite automata: strengths, weaknesses and generalizations. *Proc. 39th FOCS,* 1998, pp. 332–341. http://xxx.lanl.gov/abs/quant-ph/9802062. 340
2. Rūsiņš Freivalds. Fast probabilistic algorithms. *Lecture Notes in Computer Science,* 1979, Vol. 74, pp. 57–69. 344
3. Attila Kondacs and John Watrous. On the power of quantum finite state automata. In *Proc. 38th FOCS*, 1997, pp. 66–75. 340, 340, 340, 341
4. Christopher Moore, James P. Crutchfield Quantum automata and quantum grammars. http://xxx.lanl.gov/abs/quant-ph/9707031. 340, 340

Decomposable Bulk Synchronous Parallel Computers*

Martin Beran

Faculty of Mathematics and Physics, Charles University
Malostranské nám. 25, 118 00 Praha 1, the Czech Republic
beran@ss1000.ms.mff.cuni.cz

Abstract. The Bulk Synchronous Parallel (BSP) computer is a generally accepted realistic model of parallel computers introduced by Valiant in 1990. We present an extension to the BSP model – a decomposable BSP (dBSP for short). Performance of several elementary algorithms, namely broadcasting, prefix computation, and matrix multiplication, is analyzed on BSP and dBSP models. For a suitable setting of parameters, these algorithms run asymptotically faster on dBSP than on BSP. We also show how space-bounded sequential algorithms can be transformed into pipelined ones with bounded period on dBSP. Such a transformation is proved impossible for the BSP model. Finally, we present an algorithm for the simulation of dBSP on BSP.

1 Introduction

The bulk synchronous parallel (BSP) model of parallel computation was defined by Valiant in 1990 in [14]. A BSP computer consists of p processors with local memories, which can communicate by sending messages via a router. A computation consists of S supersteps and is periodically synchronized after each superstep. During the i-th superstep, each processor makes w_i local operations and sends or receives h_i messages[1] (such a communication request is called the h-relation). The sent messages are available at the destination processors in the beginning of the next superstep. The superstep is finished by a barrier synchronization. The computation takes time $T^{\mathrm{BSP}} = \sum_{i=1}^{S}(w_i + h_i g(p) + l(p))$. The nondecreasing functions $g(p)$ (network bandwidth per processor) and $l(p)$ (communication latency and barrier synchronization time) are machine dependent parameters defining performance of the router. An extensive research on BSP algorithms and implementation of the model on real computers has been done in recent years [2,3,4,6,9,10,12].

The standard BSP charges communication between any pair of processors equally. Thus, it cannot exploit communication locality present in many algorithms. By communication locality we mean that a processor communicates not

* This research was supported by the GA ČR grant No. 201/98/0717.

[1] w_i and h_i are both maximum over all the processors.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 349–359, 1999.

with all, but only with “close” (in some sense) other processors. Not distinguishing between communication to “short” and “long” distances yields too pessimistic estimates of time complexity in some cases. A trend to extend abstract models of parallel computers by some notion of locality can be traced in recent literature. Some examples are virtual topologies [8], Hierarchical PRAM [5], and Delay PRAM [11]. An argument – from the point of view of the first, second, and weak parallel machine classes [15,16] – for introducing some notion of locality into BSP was given in [1]. The E-BSP model [7] presents an approach how to add locality to BSP. In the present paper, we introduce yet another model allowing for locality control, namely decomposable BSP. Its main idea, partially inspired by the possibility (mentioned by Valiant [14]) of switching off the synchronization mechanism for any subset of processors in BSP and by the notion of recursively divisible problems [11], is to identify parts of a computation, when the processors work in separate groups, with no communication among the groups. A dBSP computer can dynamically partition itself into clusters which behave as dBSP computers with smaller p and therefore allow for faster communication, thanks to smaller values of $g(p)$ and $l(p)$. Processors in different clusters cannot communicate. If they need to do so, the clusters can be joined again to a single large machine. Partitioning can be done repeatedly (possibly each time into different clusters) and recursively (clusters can further split into subclusters). The dBSP model is designed so that any BSP algorithm can run on it with unchanged number of processors and time and space complexity.

The paper is organized as follows: the dBSP model is defined in Sect. 2. Then performance of some elementary algorithms on BSP and dBSP is compared in Sect. 3. Section 4 shows how a sequential algorithm with space complexity $S(n)$ can be simulated on a dBSP machine with period $O(S^k(n))$. It is proved that this bound on the period cannot be obtained on a standard BSP. In Sect. 5 we give an algorithm for simulation of dBSP on BSP. The last Sect. 6 concludes the paper and formulates some open problems and directions for further research.

2 Description of dBSP

The Decomposable Bulk Synchronous Parallel (dBSP) computer is an extension of the standard BSP computer. We denote dBSP$(p, g(p), l(p))$ a dBSP computer with p processors and particular router performance functions $g(p)$ and $l(p)$. In addition to the classical BSP computation, the processors can execute the instruction `split(i)`, $i \in \{1, \ldots, p\}$. This instruction must be executed by all the processors in the same superstep. Beginning from the next superstep, the computer is partitioned into submachines (clusters) $C_1, \ldots, C_s$, where s is the number of different values of i in the `split(i)`. Processors which specified the same i in the split instruction belong to the same submachine (i.e., to the same class of equivalence) of p_i processors. The processors in a submachine C_i are assigned the new indices $1, \ldots, p_i$, used as target addresses of messages sent in the submachine. All the submachines compute independently as separate $C_i = $ dBSP$(p_i, g(p_i), l(p_i))$ computers in the subsequent supersteps, until the processors execute the `join` instruction. All the processors of the same submachine must execute join in the same superstep. After a join, the computation of the submachine is suspended until a join is performed in all the submachines. Then, all the processors work together again as a standard BSP. In the same computation, the computer may be decomposed repeatedly and the partitioning may vary from one split to the other. The submachines can further recursively decompose, e.g. a submachine C_i can split into smaller submachines $C_{i,1}, \ldots, C_{i,s_i}$. Each join corresponds to one level of split. It means that, for example, after a sequence of $C \xrightarrow{\texttt{split}} \{C_1, C_2\} \xrightarrow{\texttt{split}_2} \{C_1, \{C_{2,1}, C_{2,2}, C_{2,3}\}\} \xrightarrow{\texttt{join}_2} \{C_1, C_2\}$, the computer is partitioned in the same way as after the first split. Processors in different submachines cannot communicate with each other. If they want to communicate, first a join must be performed. The join operation is a reversal of the corresponding split. Therefore, if a split created a set of submachines, all of them must be joined at once (no "partial joins" are allowed). The operations split and join are realized by the router as parts of the synchronization. The computation ends if all the submachines finish their work.

The time complexity for the supersteps in which the dBSP computer is not decomposed is the same as the BSP time complexity, i.e. $T_k = w_k + h_k g(p) + l(p)$ for the k-th superstep. The time between a pair of corresponding split and join is a maximum of the time complexities of computations performed by submachines: if the number of supersteps performed by the submachine i is S_i, then $T = \max_i\{T_i\} = \max_i \left\{\sum_{k=1}^{S_i}(w_{k,i} + h_{k,i} g(p_i) + l(p_i)\right\}$. The space complexity is defined as the sum of space consumed by all the processors.

We use the logarithmic cost for local computation. As usually, analysis of concrete algorithms with sufficiently bounded word size can be simplified using the uniform cost. We do not require to join the submachines back to the non-decomposed machine by the end of the computation.

3 Elementary Algorithms

In this section, we show several simple algorithms and their time complexities on BSP and dBSP. If we use a concrete function for g and l, we usually let $g(p) = l(p) = p^a$ for some $0 < a \leq 1$. These values correspond to a mesh network of dimension $1/a$. All the presented problems possess significant locality and therefore the algorithms run faster on dBSP than on BSP.

Broadcasting. The task of broadcasting is to communicate a piece of data stored in one processor's memory to all the other processors. The scheme of the algorithm [14] is a k-ary tree, $k \geq 2$. The source processor sends the data to $k-1$ other processors in one superstep. Moreover, dBSP computer splits itself into k submachines of equal size. In the following supersteps, each processor already having the data sends them to $k-1$ processors. Every dBSP submachine recursively splits into k clusters. The computation finishes after $\log_k p$ supersteps.

The time complexities are:

$$T^{\mathrm{BSP}} = ((k-1)g(p) + l(p)) \log_k p \,,$$
$$T^{\mathrm{dBSP}} = \sum_{i=1}^{\log_k p} ((k-1)g(k^i) + l(k^i)) \,.$$

For $g(p) = l(p) = p^a$, we get

$$T^{\mathrm{BSP}} = kp^a \log_k p \,,$$
$$T^{\mathrm{dBSP}} = \sum_{i=1}^{\log_k p} k^{ai+1} = k^{a+1}(p^a - 1)/(k^a - 1) \approx kp^a \,.$$

Thus, the dBSP algorithm runs faster by the speedup factor $T^{\mathrm{BSP}}/T^{\mathrm{dBSP}} = \log_k p$. The same asymptotic time bound applies also to the aggeration and prefix computation.

Matrix Multiplication. The $O(n^3)$ matrix multiplication algorithm can be parallelized in a straightforward way [10]: $n \times n$ matrices A and B are both partitioned into $\sqrt{p} \times \sqrt{p}$ equally sized square blocks. The processor $p_{i,j}$ holds the blocks $A_{i,j}$ and $B_{i,j}$ and computes the block $C_{i,j}$ of $C = A \cdot B$. The BSP computation runs in 2 supersteps. First, $p_{i,j}$ sends $A_{i,j}$ to all $p_{i,\cdot}$ and $B_{i,j}$ to $p_{\cdot,j}$. Using the received blocks, $p_{i,j}$ computes $C_{i,j} = \sum_k A_{i,k} B_{k,j}$ in the second superstep. The algorithm modified for dBSP has 5 supersteps. In the first one, the machine is partitioned so that every row of $\sqrt{p}$ processors belongs to a separate submachine. The second supersteps includes exchanging of A's blocks in rows and the join operation. Then, a similar partitioning into columns and distributing B's blocks is performed in the next 2 supersteps. Finally, blocks of C are computed. The computation runs in time

$$T^{\mathrm{BSP}} = n^3/p + 2n^2/\sqrt{p} \cdot g(p) + 2l(p) \,,$$
$$T^{\mathrm{dBSP}} = n^3/p + 2n^2/\sqrt{p} \cdot g(\sqrt{p}) + 3l(p) + 2l(\sqrt{p}) \,.$$

If we let $g(p) = l(p) = p^a$ and $p = n^b$ for some $0 < a \leq 1$ and $0 < b \leq 2$, we obtain

$$T^{\mathrm{BSP}} = n^{3-b} + 2n^{2+ab-b/2} + 2n^{ab} = \Theta(n^{2+ab-b/2} + n^{3-b}) \,,$$
$$T^{\mathrm{dBSP}} = n^{3-b} + 2n^{2+ab/2-b/2} + 3n^{ab} + 2n^{ab/2} = \Theta(n^{2+ab/2-b/2} + n^{3-b} + n^{ab}) \,.$$

For example, on a 2-dimensional mesh, $a = 1/2$, and the speedup is $T/T^{\mathrm{dBSP}} = \Omega(n^{\epsilon})$, i.e. more than a constant, for $b = 1 + \epsilon$.

4 Period of Pipelined Computation

Up to now, we have studied concrete parallel algorithms for computing a single instance of a problem. Now we turn our attention to processing a sequence of instances of the same problem. The idea of pipelined computation [16] provides a method how to transform a sequential algorithm solving single instances of the problem into an efficient parallel algorithm for solving sequences of instances. The computer successively reads individual instances[2], each of the same size, performs the computation, and outputs the results. Several instances are being processed concurrently. We denote N the number of instances in a sequence and n the size of individual instances. We assume that all time and space bounds are time and space constructible.

First, we define the complexity measures of pipelined computation which we are interested in. We also define a restriction of pipelining computation that must handle every instance separately without sharing any data among several instances.

Definition 1. *The* period $P(n)$ *of a pipelined computation is the maximum time elapsed between beginnings or – equivalently – ends of processing of two subsequent inputs in a sequence. The* time $T(n)$ *of a pipelined computation is the maximum time elapsed before the first output of the sequence is available. The* space *of a pipelined computation is the maximum amount of space needed to process a sequence of arbitrary length.*

Definition 2. *A* strictly pipelined computation *is a pipelined computation in which each instance is processed separately, i.e. a value stored during a computation of an instance cannot be used in a computation of any other instance.*

The following theorem and corollary present a way of transformation of a sequential algorithm into a parallel pipelined algorithm. The processors are logically viewed as connected to a line. Every instance successively flows through all the processors. Each processor has a buffer to store q instances. The processor performs a part of computation for every instance in its buffer and after that it sends the buffer to the next processor. The computer is repeatedly repartitioned into odd and even pairs of processors while transferring the buffers. The corollary says that if a suitable number of processors is used, the period of pipelined computation depends only on the space complexity of the original sequential algorithm.

Theorem 1. *Let $\mathcal{A}$ be a sequential algorithm working on a RAM in space $S(n) = \Omega(n)$ and time $T(n) \geq S(n)$. Let $\mathcal{M}$ be a dBSP$(p, g(p), l(p))$ computer with $g(p) = O(p)$, $l(p) = O(p)$. Then for any $1 \leq q \leq N$ a sequence of N inputs to $\mathcal{A}$ can be processed by $\mathcal{M}$ in time $O(qT(n)+pqS(n)+pl(p))$, space $O(pqS(n))$, and period $O(T(n)/p + S(n) + l(p)/q)$ (under uniform cost).*

Proof. The processors are seen as though they were connected into a linear list. For each processor except the last one, there exists its unique successor. Each of p processors holds q instances in a buffer, computes $T(n)/p$ steps of each instance, then sends them at once to the next processor and receives the next q instances

[2] Only the first n processors have access to the input.

from the previous processor. The communication is done while the computer is partitioned into pairs of successive processors (a repartitioning into odd and even pairs is necessary). Three supersteps are needed to perform q periods. The following formula holds for the period:

$$qP(n) = q \cdot \frac{T(n)}{p} + 2qS(n) \cdot g(2) + 2l(2) + l(p)\,,$$
$$P(n) = \frac{T(n)}{p} + 2S(n).g(2) + \frac{2l(2)}{q} + \frac{l(p)}{q} = O\left(\frac{T(n)}{p} + S(n) + \frac{l(p)}{q}\right)\,.$$

Filling p buffers of p instances takes $O(pqP(n)) = O(qT(n) + pqS(n) + pl(p))$ time steps and space $O(pqS(n))$. □

Corollary 1. *Let $\mathcal{A}$ be a sequential algorithm working on RAM in space $S(n) = \Omega(n)$ and time $T(n) \geq S(n)$. Let $\mathcal{M}$ be a dBSP$(p, g(p), l(p))$ computer with $p = T(n)/S(n)$ processors and $g(p) = O(p)$, $l(p) = O(p)$. Then a sequence of inputs to $\mathcal{A}$ can be processed by $\mathcal{M}$ in time $O(T^2(n))$, space $O(T^2(n)/S(n))$, and period $O(S^2(n))$ (under logarithmic cost).*

Proof. We use Theorem 1 and let $p = q = T(n)/S(n)$. This gives us period $P(n) = O(S(n))$, time and space $O(T^2(n)/S(n))$ under uniform cost. As we use the logarithmic cost, we multiply time and period by logarithm of space per processor $\log T(n) \leq \log(c^{S(n)}) = O(S(n))$ (for some constant c). □

The previous theorem naturally raises a question whether the possibility of partitioning of dBSP is necessary to get the above upper bounds on a period. In other words, could the same period, i.e., $O(S^2(n))$, be achieved on a BSP computer? The next theorem gives a negative answer. Not only the algorithms from Theorem 1, but also any strictly pipelined algorithm on a BSP computer with polynomially bounded $g(p)$, $l(p)$, has a period larger than $\Omega(S^k(n))$, for any constant $k > 0$. The difference between BSP and dBSP follows from the fact that a BSP computer with many processors needs a long time to perform some communication while a dBSP computer with the same parameters, but suitably partitioned, can do it faster.

Theorem 2. *Let $\mathcal{M}$ be a BSP$(p, g(p), l(p))$ computer with $g(p), l(p) = \Omega(p^a)$ for some constant $a > 0$. Let $\mathcal{P}$ be a problem such that the following condition holds for the time $T(n)$ and space $S(n)$ complexity of the fastest algorithm solving $\mathcal{P}$[3]: $\forall b > 0\colon T(n) \geq \omega(S^b(n))$. Then there is no strictly pipelined algorithm $\mathcal{A}$ for $\mathcal{P}$ and no constant $k > 0$ such that $\mathcal{A}$ runs on $\mathcal{M}$ with period $O(S^k(n))$.*

Proof. BSP time of the pipelined computation of N instances of size n is $T^{\mathrm{BSP}} = t(n) + (N-1)P(n)$. The inequation $NT(n) \leq p.(t(n) - P(n) + NP(n))$ holds

[3] $f(n) = \omega(g(n)) \overset{\text{def}}{\iff} \lim_{n\to\infty} g(n)/f(n) = 0$.

because the sequential time is the best possible[4]. Let us assume $\exists k > 0\colon P(n) = O(S^k(n))$, i.e. $\exists c, k > 0\colon P(n) \le cS^k(n)$. Then

$$p \ge \lim_{N\to\infty} \frac{NT(n)}{t(n) - P(n) + NP(n)} = \frac{T(n)}{P(n)} \ge \frac{T(n)}{cS^k(n)}\,.$$

We may assume that computation of each of N instances of $\mathcal{P}$ involves at least one communication operation in every input processor, hence $\exists d > 0\colon Ndp^a \le Ng(p) \le T^{\mathrm{BSP}} = t(n) + (N-1)P(n)$ and

$$P(n) \ge \lim_{N\to\infty} \frac{Ndp^a - t(n)}{N-1} = dp^a \ge d \cdot \left(\frac{T(n)}{cS^k(n)}\right)^a\,.$$

From the relation between $T(n)$ and $S(n)$ we get $\forall k > 0\colon P(n) = \omega(S^k(n))$, which is a contradiction with the assumption $P(n) \le cS^k(n)$. □

The theorem says that dBSP is strictly faster than BSP in some computations. The following theorem gives a lower bound on the period for dBSP, stating that a constant period cannot be reached by a dBSP computer. It is caused by a necessity to either have large clusters, or to perform repartitioning frequently to support communication between processors in different clusters.

Theorem 3. *Let $\mathcal{M}$ be a dBSP$(p, g(p), l(p))$ computer with $g(p), l(p) = \omega(1)$. Let $\mathcal{P}$ be a problem with the time and space complexity $T(n) \ge S(n) \ge \omega(1)$. Then there is no strictly pipelined dBSP algorithm for $\mathcal{P}$ running with a constant period.*

Proof (sketch). Because we use the logarithmic time cost, only a constant number of bits can be processed during a period in one processor. Therefore, after a constant number of steps, a communication in a cluster of non-constant size must occur. Such a communication takes time $\omega(1)$ which makes the average time of one step $\omega(1)$. □

5 A BSP Simulation of dBSP

In this section, we study how BSP and dBSP computers can simulate each other. An immediate observation is that any BSP algorithm can be directly executed by a dBSP machine, because the dBSP model is an extension to the BSP. The reverse simulation is a bit complicated.

For simplicity, we will use the uniform cost in the following simulation of dBSP on BSP. The logarithmic cost increases all the results by a multiplicative factor of $\log p$. We assume that the communication cost function $g(p)$ is non-decreasing and fulfils the inequations $g(kp) \le kg(p)$ and $g(p+k) \le g(p) + g(k)$. The same conditions hold for $l(p)$. Each dBSP processor is simulated by the

[4] The BSP algorithm is strictly pipelined, thus no precomputed values can be used for more than one instance.

corresponding BSP processor. In the worst case, the dBSP computer is divided into submachines of a constant size. Therefore an h-relation can be routed in time $O(h)$, while the simulating BSP needs the time $O(hg(p)+l(p))$. This gives the slowdown factor of $O(g(p)+l(p))$. The main technical difficulty is caused by the fact that in the dBSP, message destinations are determined using processor indices local to a cluster, while the BSP uses only global indices of processors. Therefore, we must maintain a mapping from local to global indices and update it after each split and join operation. The mapping is implemented by routing each message through an intermediate processor. The computation of routes involves sorting and prefix sum computation. This causes a slower simulation of split and join operations. By a proper analysis, we get the following theorem.

Theorem 4. *A BSP$(p, g(p), l(p))$ computer can simulate a dBSP$(p, g(p), l(p))$ algorithm within an additional space $O(p^2)$. The simulation time complexity is as follows:*

1. *A superstep performed on all clusters of a partitioned dBSP, which takes T_i^{dBSP} in the cluster i, is simulated in time $T^{\mathrm{BSP}} \leq O\big(\max_i\{T_i^{\mathrm{dBSP}}\} \cdot \max\{g(p), l(p)\}\big)$.*
2. `split` *and* `join` *both take time $O((1+g(p)+l(p))\log p)$.*

Proof.

1. Let us assume that translating the local index in a cluster of a destination processor of a message into the global index of the processor in the BSP machine introduces only a constant-factor slowdown. Then the following formulas hold for the cluster i containing p_i processors:

$$\begin{aligned} T_i^{\mathrm{dBSP}} &= w_i + h_i g(p_i) + l(p_i) \geq w_i + (h_i+1)\min\{g(p_i), l(p_i)\}\,, \\ T_i^{\mathrm{BSP}} &= w_i + h_i g(p) + l(p) \leq w_i + (h_i+1)\max\{g(p), l(p)\}\,, \\ \frac{T_i^{\mathrm{BSP}}}{T_i^{\mathrm{dBSP}}} &\leq \frac{w_i + (h_i+1)\max\{g(p), l(p)\}}{w_i + (h_i+1)\min\{g(p_i), l(p_i)\}} \leq \frac{\max\{g(p), l(p)\}}{\min\{g(p_i), l(p_i)\}}\,. \end{aligned}$$

   Time of the superstep is the maximum over all the clusters, i.e.

$$T^{\mathrm{BSP}} \leq \max_i\{T_i^{\mathrm{BSP}}\} \leq \max\left\{T_i^{\mathrm{dBSP}} \cdot \frac{\max\{g(p), l(p)\}}{\min\{g(p_i), l(p_i)\}}\right\}\,.$$

   The smallest meaningful cluster has 2 processors. This fact gives the upper bound stated in the theorem.
2. During splitting and joining we must ensure validity of the assumption from the first part of the proof. A global enumeration of clusters is maintained (cluster numbers are $0, \ldots, p-1$, in the beginning of computation, all the processors belong to the cluster 0). During a `split` superstep[5], each processor generates a tuple $\langle g_i, c_i, i\rangle$, where i is the current index of the processor, g_i is the current global cluster index, and c_i is the new subcluster index

[5] Let us suppose `split` is performed in all the clusters simultaneously.

of this processor. The tuples are sorted in the lexicographic order. Using a prefix sum algorithm, a new global enumeration of clusters and ranks of processors in clusters is computed. A resultant tuple $\langle g'_i, c_i, i, r_i, j\rangle$ is sent to the processor i. g'_i is the new global cluster index, r_i is the rank of the processor in the new cluster, and j is the index of the processor which has got the processor i tuple after sorting. A message from r_i to r_k (source and destination addresses in a cluster) is first sent from i to $j + (r_k - r_i)$. This processor knows the value of k and can send the message to its final destination.
Sorting [13] takes time $O((1+g(p)+l(p))\cdot\log p)$, prefix sums and detection of `join` instruction can be also done in this time bound which yields the stated time upper bound. Each processor needs a stack of $O(p)$ tuples, because splits can be recursively nested to depth p. □

6 Conclusion

We have defined a new class of parallel computer models based on the BSP model, namely the decomposable BSP, which allows to exploit communication locality in BSP-like computations. This model is an extension to the standard BSP. Therefore any standard BSP algorithm can run on it with the complexity unchanged. If we are able to identify parts of a BSP algorithm, in which subsets of processors compute independently on each other, a dBSP machine allows to improve performance of the communication part of the algorithm. We have analyzed several dBSP algorithms which achieve a speedup in comparison to their BSP counterparts. Then we have focused to algorithms for pipelined processing of instances of the same problem. We have presented a method how to obtain a period polynomially related to sequential space on a dBSP with sufficiently many processors. We have proved that a dBSP pipelined algorithm can gain a shorter period than any BSP algorithm. Finally, we have presented mutual simulations of BSP and dBSP computers. According to the results of the previous sections, the dBSP model seems to be a viable extension to the BSP.

Although this paper presents basic results regarding the decomposable BSP, many new problems emerged during its preparation. They will be addressed by further research. Following is a non-exhaustive list of open questions which are to be answered.

- Find more problems which can be solved on a dBSP more effectively than on a BSP.
- Is there a concise characterization of problems efficiently solvable on the dBSP model? How such problems relate to the classes of ideally and recursively divisible problems, as defined in [11]?
- Could membership of dBSP in the class of weak parallel machines [16] be established? It means that we aim to a proof of a reversal of Corollary 1. Given a pipelined algorithm for a problem $\mathcal{P}$ with period $P(n)$, we want to obtain a sequential algorithm for $\mathcal{P}$ with space complexity $O(P^k(n))$ for some constant $k > 0$.

- Can we prove a lower bound higher than $\omega(1)$ for the period of a strictly pipelined computation on a dBSP computer?
- Improve the simulations or prove their optimality.
- The dBSP model can be further extended. An m-dBSP has p processors and m routers. Each router is connected to all the processors and can be partitioned independently on the other routers. Therefore two processors can belong to different clusters of one router and still be connected via another router. What are the characteristics of the m-dBSP model in comparison to the dBSP and BSP?
- Values of g and l in a cluster depend only on the number of processors in the cluster. How would the model behave if we defined a distance $d_{i,j}$ between each pair of processors (i,j) and g, l for a cluster C were dependent on $\max_{(i,j)}\{d_{i,j}\}$, for $i,j \in C$?

References

1. Martin Beran. *Computational Power of BSP Computers.* In Proceedings of SOFSEM '98, volume 1521 of Lecture Notes in Computer Science, pages 285–293. Springer-Verlag, 1998. 350
2. Alexandros V. Gerbessiotis and Constantinos J. Siniolakis. *Primitive Operations on the BSP Model.* Technical Report PRG-TR-23-96, Oxford University Computing Laboratory, Oxford, October 1996. 349
3. Alexandros V. Gerbessiotis and Leslie G. Valiant. *Direct Bulk-Synchronous Parallel Algorithms.* Journal of Parallel and Distributed Computing, 22:251–267, 1994. 349
4. Mark Goudreau, Kevin Lang, Satish Rao, Torsten Suel, and Thanasis Tsantilas. *Towards Efficiency and Portability: Programming with the BSP Model.* In SPAA '96: Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures, pages 1–12. ACM Press, 1996. 349
5. T. Heywood and S. Ranka. *A Practical Hierarchical Model of Parallel Computation: I and II.* Journal of Parallel and Distributed Computing, 16:212–249, November 1992. 350
6. B. H. H. Juurlink and H. A. G. Wijshoff. *Communication Primitives for BSP Computers.* Information Processing Letters, 58:303–310, 1996. 349
7. Ben H. H. Juurlink and Harry A. G. Wijshoff. *The E-BSP Model: Incorporating General Locality and Unbalanced Communication into the BSP Model.* In Proceedings of Euro-Par'96 (Vol. II), Volume 1124 of Lecture Notes in Computer Science. Springer-Verlag, 1996. 350
8. Klaus-Jörn Lange. *On the Distributed Realization of Parallel Algorithms.* In Proc. of SOFSEM '97, Volume 1338 of Lecture Notes in Computer Science, pages 37–52. Springer-Verlag, 1997. 350
9. W. F. McColl. *Bulk Synchronous Parallel Computing.* In John R. Davy and Peter M. Dew, editors, Abstract Machine Models for Highly Parallel Computers, pages 41–63. Oxford University Press, 1995. 349
10. W. F. McColl. *Scalable Computing.* Lecture Notes in Computer Science, 1000:46–61, 1995. 349, 352
11. Rolf Niedermeier. *Towards Realistic and Simple Models of Parallel Computation.* PhD thesis, Fakultät für Informatik, Eberhard-Karls=Universität Tübingen, Tübingen, 1996. http://www-fs.informatik.uni-tuebingen.de/ niedermr/publications/di2.ps.Z 350, 350, 357

12. *Oxford BSP research group*. http://www.comlab.ox.ac.uk/oucl/groups/bsp/ 349
13. Constantinos J. Siniolakis. *On the Complexity of BSP Sorting*. Technical Report PRG-TR-09-96, Oxford University Computing Laboratory, 1996. http://www.comlab.ox.ac.uk/oucl/users/constantinos.siniolakis/index.html 357
14. Leslie G. Valiant. *A Bridging Model for Parallel Computation*. Communications of the ACM, 33(8):103–111, 1990. 349, 350, 351
15. P. van Emde Boas. *Machine Models and Simulations*. Handbook of Theoretical Computer Science, A:1–66, 1990. 350
16. Jiří Wiedermann. *Weak Parallel Machines: A New Class of Physically Feasible Parallel Machine Models*. In I. M. Havel and V. Koubek, editors, Mathematical Foundations of Computer Science 1992, 17th Int. Symposium (MFCS'92), Volume 629 of Lecture Notes in Computer Science, pages 95–111, Berlin, 1992. Springer-Verlag. 350, 353, 357

Component Change and Version Identification in SOFA*

Přemysl Brada

[1] Department of Computer Science and Engineering
University of West Bohemia
Univerzitní 22, 30614 Plzeň, Czech Republic
brada@kiv.zcu.cz
www-kiv.zcu.cz/

[2] Department of Software Engineering, Charles University,
Prague, Czech Republic
http://nenya.ms.mff.cuni.cz/

Abstract. In the area of component software, the work so far has concentrated primarily on the key issues of architecture specification and component updating. However, the problems of maintaining application consistency and versioning of components as they evolve have received less attention, and the available solutions are either ad-hoc or not well suited for component applications. In this paper we evaluate these solutions and present a new approach to component versioning developed for the SOFA architecture. Based on the analysis of changes between two versions, component revision numbers and change indications are derived as a lightweight compatibility description which is subsequently used for consistency verification during application composition or component update. Thus it is shown how giving the revision numbers a precise semantics and providing additional versioning information provides a support for the tasks of configuration management in component applications.

1 Introduction

In the last few years, research as well as practical applications of software component technology became a rapidly growing field. Research in the area is abundant [1,5,4,2] and several commercial systems are available or emerging [7,9,8]. While the important architectural issues such as component description languages, connectors and even composition of component-based applications are extensively researched [4,2,11], software configuration management (SCM) issues related to component technology have received less attention.

As in 'ordinary' software systems, the role of SCM in the component world is to help identifying and creating consistent configurations. In this paper we take

* This work was partially supported by the Grant Agency of the Czech Republic — project 201/99/0244.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 360–368, 1999.

the view that because revision numbers are used to denote the steps of a software component evolution they implicitly express its backward compatibility and could thus help in consistency checks when a component is being upgraded within an application. The problem is that mostly there are no precise rules for the creation and interpretation of revision numbers. While this is tolerable as long as they are only used and interpreted by humans, it is insufficient for component-based applications where the application should ideally be composed by the *user* (not the developer) with the help of automated tools.

Note: As the term "component" is used to denote many different things in software engineering and research [1], for the purpose of this paper it is defined as black box re-usable piece of software distributed and used in binary form, with defined interfaces (provided and required) and optionally behaviour (semantics). Applications are composed from components using hierarchical aggregation, connectors, or a combination of both.

This paper describes the system for component version and compatibility identification in the SOFA/DCUP (SOFtware Appliances, Dynamic Component UPdating) framework [5], where it is used in component installation and subsequent updates. The rest of this section briefly introduces SOFA principles and architecture. Section 2 describes how the version and compatibility identification is derived, expressed and used in SOFA. A discussion of related work including the approaches to checking compatibility and consistency is in Section 2.3. The paper ends with remarks on future research directions and a conclusion.

1.1 Overview of SOFA

Software components in SOFA/DCUP [5] are black/grey-box objects with defined interfaces and behaviour. An interface is a named set of method signatures to which a behaviour description (a *protocol* [6]) is attached. Using regular-like expressions, it describes the allowed sequencing of interface method calls. The black-box view of a component is called *component frame* and defines the set of interfaces the component will provide to its clients and require from its environment, plus its behaviour protocol (see Figure 1). A component's *architecture* creates the gray-box view by defining, on the first level of frame nesting, which subcomponents will constitute its implementation. An application is formed by hierarchical nesting of components.

Interface and component frame specifications are written in the SOFA Component Description Language (CDL), similarly to other systems [3,16,8]. The CDL descriptions of provided interfaces P, required interfaces R, and behavioural protocol B are called *specification levels* in this paper (the same notation is used for interfaces, where P denotes the method signatures and R is omitted).

SOFAnet, the infrastructure which supports component development and deployment, consists of a network of co-operating nodes. Each SOFAnode contains a central template repository (TR) which stores component frames plus implementations (e.g. in the form of Java package files) as well as their versioning information. Other parts of the node provide various support operations: the Run part is used for launching and running component applications and includes the DCUP interface for run-time updates, and the Made part creates an environment for developing new components.

```
interface IDBServer {
  void insert(in int key, in tdata data);
  void delete(in int key);
  void query(in tdata mask, out tdata data);
protocol:
  ( insert + delete + query )*    // ‘+’ means ‘either-or’
};                                // ‘*’ denotes repetition
    ...
frame Database {
  provides:      IDBServer dbSrv;
  requires:      IPlainLogging  dbLog;
  protocol:
    !dbLog.init ;     // outgoing call delegated to the dbLog interface
    ( ?dbSrv.insert { !dbLog.log } +   // an incoming call via dbSrv
      ?dbSrv.delete { !dbLog.log } +   // leads to an outgoing on dbLog
      ?dbSrv.query )*
}
```

Fig. 1. SOFA interface and component frame CDL specification

Versioning in SOFA is based on intensional versioning [12] and implemented via specialised versioning objects stored in the TR. These objects hold both the revision and variant information, one object per each version of a component.

2 SOFA Component Compatibility and Versioning

Despite various attempts described in Section 3, there is no standardised way to classify changes to software units which would support analyses of their compatibility. Such analyses are however necessary for the automation of (component) updates that preserve configuration consistency. In this section, we describe a system developed for the SOFA framework in an attempt to grasp the compatibility and versioning problems clearly and find a working solution.

The general definition of component compatibility used here is as follows: for a component C, its version V_2 is *backward compatible* with a previous version V_1 if and only if (1) all clients of V_1 can use V_2 without adaptation and (2) V_2 can function in all execution environments of V_1 without adaptation. This notion of compatibility takes into account the dependencies of the component in its operating environment, an aspect neglected in some systems [16,15].

To form a ground for the versioning system we first develop a classification of changes between two revisions of a component based on analysing the differences in their CDL specification levels. Let $L_1, L_2 \in \{P, B, R\}$ denote the same specification level in two frame revisions (e.g. the sets of provided interfaces, or the frame protocols). Let $F(L_i)$ be the set of all component frames which can be denoted by L_i. When comparing L_1 and L_2, we say that

- L_2 *exactly matches* L_1 (denoted $L_2 = L_1$) iff $F(L_1) = F(L_2)$;
- L_2 is a *specialisation* of L_1 (denoted $L_2 \supset L_1$) iff $F(L_2) \subset F(L_1)$;
- L_2 is a *generalisation* of L_1 (denoted $L_2 \subset L_1$) iff $F(L_2) \supset F(L_1)$;
- L_2 is a *mutation* of (incomparable to) L_1 (denoted $L_2 \triangle L_1$) iff $F(L_2) \not\supset F(L_1) \wedge F(L_2) \not\subseteq F(L_1)$.

This definition of $\subset$ on P, B, and R corresponds to the usual understanding (the specialised version has more features and denotes fewer objects) and moreover unifies the comparison of different types of specifications. Thus we can use the same classification of changes for other types of specifications as well (e.g. current interfaces functionality or future component dynamics descriptions). In practice the comparison is based directly on the CDL specification — for provisions and requirements as a set operation on the union of interface method signatures, for protocols as a language inclusion operation.

2.1 Determining Component Compatibility

Based on the specification changes, we classify component[1] compatibility as follows (P_1 and P_2 denote the provided interfaces of two revisions V_1 and V_2 of one component, similarly R_i and B_i). V_2 is with respect to V_1:

- *equal*, iff $P_2 = P_1 \wedge R_2 = R_1 \wedge B_2 = B_1$;
- *(strongly) compatible*, iff $P_2 \supseteq P_1 \wedge B_2 \subseteq B_1 \wedge R_2 \subseteq R_1$;
- *conditionally compatible*, iff $P_2 \supseteq P_1 \wedge ((R_2 \supseteq R_1 \vee R_2 \triangle R_1) \vee (B_2 \supseteq B_1 \vee B_2 \triangle B_1))$;
- *incompatible*, iff $P_2 \subset P_1 \vee P_2 \triangle P_1$.

The *strongly compatible* version V_2 is a straightforward replacement for V_1 (variant properties allowing) because it provides more, and has less strict behaviour and requirements (a 'contravariant' change). The new revision is *incompatible* if the provided interface was generalised (reduced) or mutated — an adaptor has to mediate access for clients designed for the old version.

The *conditionally compatible* version has specialised or mutated requirements and/or behaviour (a 'covariant' change). From a purely subtyping view, mutation changes on any level constitute subtype incompatibility (e.g. if the `Database` component from Figure 1 changed its requirements from the `IPlainLogging` interface to a `IUserDefLog` interface). However, in practice it is possible that the environment in which V_2 will be deployed will provide for its requirements (e.g. via a component with the `IUserDefLog` interface) or behaviour (clients accepting the new protocol). The definitions were therefore relaxed by moving the 'context-dependent' incompatibilities into the *conditional* category.

[1] Interface compatibility categories and their defining rules are similar except the R level is missing because interfaces have no dependencies.

On the other hand, if we allowed to mutate the provided interface for similar reasons the communication with current clients could break, resulting in a loss of functionality. Furthermore, the interface might be eventually changed completely while pretending (by the name) that it is the same type. As both of these effects are clearly undesirable, the SOFA versioning support forces the developer to rename the frame if an incompatible change occurs.

2.2 Revision Numbers and Change Indications

SOFA interface and frame versioning uses the change analysis described above.[2] Its results are captured in two complementary data structures: the hierarchical revision number (part of the CDL so that both the developers and the tools can check what has changed), and an indication of the types of changes at each level (stored in component version data as a more specific information for the tools).

SOFA *revision number* is defined for interfaces as an ordered pair (P, B), for component frames as an ordered triple (P, B, R) of natural numbers. A change from the previous revision in a specification level L is homomorfically mapped into an increase of the corresponding part of the revision number. That is, if for example the `IDBServer` interface is evolved by adding a signature for a method, its revision number part P is incremented (see Figure 2).

In other words, the revision number's hierarchical structure is directly derived from the levels of software component specification which in turn have relation to the types of changes. This contrasts with the usual practice of arbitrarily choosing a *Major.minor* scheme with semantics defined by rules of thumb.

```
interface IDBServer rev 1.1 {
  void insert(in int key, in tdata data);
  void delete(in int key);
  void query(in tdata mask, out tdata data);
  protocol:    ( insert + delete + query )*
};
interface IDBServer rev 2.2 {
  void set(in dbProperties prop);
  void insert(in int key, in tdata data);
  void delete(in int key);
  void query(in tdata mask, out tdata data);
  protocol:    set ; ( insert + delete + query + set )*
};
```

Fig. 2. Two consecutive revisions of the `IDBServer` interface

The *change type indication* is an ordered pair (p, b) for interfaces resp. an ordered triple (p, b, r) for frames, where $p, b, r \in \{match, gen, spec, mutation\}$.

[2] Technically the analysis is done on the component's release into SOFAnode's Template Repository and implemented using comparison of derivation trees.

Its value for two consecutive revisions is computed as described at the beginning of this section. For pairs of non-consecutive revisions it is computed as the maximum value over each pair of intermediate revisions. In the example above, the interface change type indication for revision 2.2 will be (*spec*, *mutation*) because the set of interface methods is extended and the new protocol is incomparable to the old one.

The compatibility of the whole frame is determined using the definition in Section 2.1, based on the change type values for individual levels. When components are nested, all changes in the nested components are manifested only in the parent component's architecture. This makes the approach independent on the scale of the composition.

2.3 Examples of Use

Revision and change identification are used in several operations in the SOFA framework. The revision number helps explicitly denote version dependencies — the `requires` section contains the revisions of interfaces needed for correct operation. Using the compatibility rules, the requirement for `IDBServer rev 2.2` in the example below can be satisfied by revision 2.2 but possibly also 2.5 of the interface (because they differ only in the protocol).

```
frame Database rev 2.1.0 {
  provides:      IDBServer rev 2.2  dbSrv;
  requires:      IPlainLogging rev 1.0  dbLog;
  protocol:          // omitted for brevity
}
```

Thanks to the exact meaning of (differences in) revision numbers and change types it is possible to determine, before the new version replaces the current one, if the new configuration can be consistent. For example, suppose an application currently contains revision 1.0.0 of the `Database` component, revision 2.1.0 is available as an update, and (*spec*, *spec*, *match*) is the corresponding change type indication. The difference between the revision numbers (whose 'value' is $(1, 1, 0)$) indicates that the provisions and behaviour have changed, and from the change type indications we see it was a 'covariant' change.

SOFAnode determines from this data that the new revision is conditionally compatible with the old one and can replace it only if its frame protocol conforms to that of 1.0.0; thus only the protocol conformance checks will need to be run. Most of this work can be done without analysing the full CDL source (which may be a computationally intensive task) and without the need to shut down the application – this is particularly important for high-availability systems.

3 Related Work

Ensuring that consistent software configuration is composed from its constituent units as well as classifying changes in software units and their impact on compatibility are the subject of several academic research projects as well as commercial systems. However, not always there is an explicit relation to versioning.

The work of Perry [13,14] on software unit compatibility and its impact on architecture consistency is rigorous and clearly put in the SCM context, but no link is made to versioning as such. Also, the consistency checks are always done on full function specification which may be impractical for larger systems.

Relating component and total versioning, the Ragnarok paper [11] mentions the need to reflect component changes in the architecture version identification. However, this identification is described without detail and the types of architecture changes are not classified.

Research on the C2 system at University of California [4] is concerned with static and run-time modifications to the application architecture. The authors assert that modification constraints need to be verified to ensure configuration consistency but do not provide any details about relevant methods. The versioning approach presented in this paper can thus be seen as one such method suitable for coarse-grained components.

In the Java language system [15], compatibility checks are done partly at link- and partly at run-time when inheritance or interface conformance problems are solved by exception handling. The JDK version 1.2 also defines the package version identification as a triple $(major, minor, micro)$ of natural numbers. However the meaning of these numbers lacks a precise definition, namely any relation to the compatible changes defined in the language specification.

The DCE (Distributed Computing Environment) [16] defines precisely what server interface changes are (in)compatible and how they are reflected in the interface revision ID. Clients using an interface with an incompatible version number may be refused by the server. This is a well engineered system but the compatibility rules are too rigid, and the fine granularity (interfaces only) and lack of behavioural specification make it insufficient for component versioning.

4 Open Issues and Future Research

The work on SOFA component revisioning so far has resulted in the specification of its principles and algorithms. At the time of writing, a prototype implementation of the system is being developed. Its main functions will be to perform the change analysis and generate the revision numbers and change type indications into the CDL source and SOFA node data structures, and to check compatibility based on the version information and CDL sources.

The proposed model has obviously some weaknesses and open issues as well. First, due to the method of computing revision numbers it is not possible to identify branches, i.e. alternative but backward compatible versions of an interface or component frame specification. The proposed solution to this problem is that if a branch is created (because there already exists a subsequent trunk revision) it would be required to have a different name but could specify the name and revision from which it was derived.

Also, the notion of shared dependencies as defined in [14] (in our case, formats of data created by the component etc.) is not considered in the current compatibility definitions. This may lead to run-time problems undetected during the checks preceding component update.

Future research in this project will therefore focus on these issues, mainly how to include data format dependencies in the revision and change type indication. Generalisation of the branching problem is also the need to specify compatibility of interfaces and components developed by different providers, as provider identification is part of the name.

5 Conclusion

The novel software component versioning approach presented in this paper creates 'semantically rich' software component revision numbers and change identifications based on the analysis of component CDL specification. Using the revision and change type information as a first sentry, and the full component CDL description for complete checks, configuration consistency of the prospective update can be detected automatically without the need to shut down the (possibly running) application. At the same time the ascending sequence of revision numbers conveys the traditional meaning of the historical order of component development.

Future work on the system includes solving the open issues of branching and data compatibility, as well as developing a working implementation. In a wider context, a modification of this versioning approach could be used for other systems, e.g. DCE and various Unix package deployment systems.

References

1. Szyperski, C.: Component Software. ACM Press, Addison-Wesley 1998. 360, 361
2. Allen, R., Garlan, D.: Specifying Dynamism in Software Architectures. Proceedings of Foundations of Component-based Systems Workshop, 1997. 360, 360
3. Magee, J., et al: Specifying Distributed Software Architectures. Proceedings of ESEC'95, Barcelona, Spain. 361
4. Oreizy, P.: Issues in the Runtime Modification of Software Architectures. TR-96-35. University of Carolina, Irvine, 1996. 360, 360, 366
5. Plášil, F., Bálek, D., Janeček, R.: SOFA/DCUP: Architecture for Component Trading and Dynamic Updating. Proceedings of ICCDS 98, Annapolis, Maryland, USA. 360, 361, 361
6. Plášil, F., Višňovský, S., Bešta, M.: Behavior Protocols and Components. Proceedings of TOOLS USA '99, Santa Barbara, CA, August 1999. 361
7. JavaSoft: JavaBeans 1.0 Specification. http://www.javasoft.com/beans/spec.html. 360
8. CORBA Components, Joint Revised Submisstion. OMG orbos/99-02-05. 360, 361
9. Rogerson, D.: Inside COM. Microsoft Press 1997. 360
10. IEEE Standard 1042-1987: Guide to Software Configuration Management. IEEE 1994.
11. Christensen, H.B.: Experiences with Architectural Software Configuration Management in Ragnarok. Proceedings of SCM-8 Workshop, ECOOP 1998. Springer-Verlag 1998. 360, 366

12. Conradi, R., Westfechtel, B.: Configuring Versioned Software Products. Proceedings of SCM-6 Workshop, ICSE'96, Berlin, Germany. LNCS, Springer-Verlag 1996. 362
13. Perry, D. E.: Version Control in the Inscape Environment. Proceedings of ICSE'87, Monterey, CA. 366
14. Perry, D. E.: System Compositions and Shared Dependencies. Proceedings of SCM-6 Workshop, ICSE'96, Berlin, Germany. LNCS, Springer-Verlag 1996. 366, 366
15. JavaSoft: The Java Product Versioning Specification. http://www.javasoft.com/docs/jdk1.2/docs/guide/versioning/ 362, 366
16. Peterson, M. T.: DCE: A Guide to Developing Portable Applications. McGraw-Hill, 1995. 361, 362, 366

Pattern Equations and Equations with Stuttering*

Ivana Černá[1], Ondřej Klíma[2], and Jiří Srba[1]

[1] Faculty of Informatics MU,
Botanická 68a, 602 00 Brno, Czech Republic,
{cerna,srba}@fi.muni.cz
[2] Faculty of Science MU, Dept. of Mathematics,
Janáčkovo nám. 2a, 662 95 Brno, Czech Republic,
klima@math.muni.cz

Abstract. Word equation in a special form $X = A$, where X is a sequence of variables and A is a sequence of constants, is considered. The problem whether $X = A$ has a solution over a free monoid (PATTERN-EQUATION problem) is shown to be NP-complete. It is also shown that disjunction of a special type equation systems and conjunction of the general ones can be eliminated. Finally, the case of stuttering equations where the word identity is read modulo $x^2 = x$ is mentioned.

1 Introduction

In computer science many natural problems lead to solving equations. It is the main topic in several fields such as logic programming and automated theorem proving where especially unification plays a very important role. A number of problems also exploit semantic unification, which is in fact solving word equations in some variety. A very famous result by Makanin (see [11]) shows that the question whether an equation over a free monoid has a solution is decidable. It can be even generalized in the way that existential first-order theory of equations over free monoid is decidable. Moreover adding regular constraints on the variables (i.e. predicates of the form $x \in L$ where L is a regular language) preserves decidability [13].

In this paper we consider a very practical issue of a certain subclass of equations which we call pattern equations. Many problems such as pattern matching and speech recognition/synthesis lead to this kind of equations where we consider on the lefthand side just variables and on the righthand side only constants. This work has been mostly inspired by the papers [5] and [6] where the basic approach – syllable-based speech synthesis – is in assigning prosody attributes to a given text and segmentation into syllable segments. This problem can be modelled by pattern equations over free monoid resp. idempotent semigroup and is trivially decidable. However, we could ask whether a *polynomial* algorithm exists

* The paper is supported by the Grant Agency of the Czech Republic, grant No. 201/97/0456 and by grant FRVŠ 409/1999

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 369–378, 1999.

to find a solution. Unfortunately, this problem appears intractable (supposing P $\neq$ NP) since we prove that it is NP-complete. One of the ways how to solve the problem is to use heuristic algorithms. They are the current field of interest in speech synthesis. Another approach that could be used for solving the problem is *Concurrent Constraint Programming*. For the background see [1].

We may also ask whether for a system of pattern equations (connected by conjunction resp. disjunction) exists a single equation preserving satisfiability and/or solutions. In the positive case a question of the transformation complexity arises. If the transformation can be done effectively (e.g. in linear time as it is shown in Section 4), we can concentrate on finding heuristics just for a single pattern equation where the situation could be easier to manage. The elimination of conjunction resp. disjunction is generally possible [13]. What we show is that we can find an equation preserving solutions of the system (and thus also satisfiability) which is again of our special type, i.e. it is a pattern equation. We demonstrate that for conjunction no extention of the constant and variable alphabet is necessary and the length growths polynomially where the degree of the polynomial depends on the number of equations. For the practical purposes it is much more convenient to add some new symbols into the alphabet and thus achieve just a linear space extention, which is also manifested in our paper. Similar results are formulated for disjunction.

We also examine the solvability of the equations in the variety of idempotent semigroups (bands) which we call stuttering equations. Their name comes from practical motivations. For example in the speech recognition the speaker sometimes stutters some words and we would like to eliminate this effect and enable the correct variables assignation even in the case of stuttering. Therefore we allow to eliminate multiple occurrences of the same constant into only one occurrence, which can be modelled by the identity $x^2 = x$.

Local finiteness of free bands yields decidability of the satisfiability problem even in the general case and we give an exponential upper bound on the length of any solution up to band identities. A polynomial time decision procedure for the word problem in idempotent semigroups is straightforward and can be found in the full version of this paper [2]. Consequently, the satisfiability problem of stuttering equations belongs to NEXPTIME. The complexity issues for stuttering *pattern* equations are also discussed.

2 Basic Definitions

Let C be a finite set of *constants* and V be a finite set of *variables* such that $C \cap V = \emptyset$. A *word equation* $L = R$ is a pair $(L, R) \in (C \cup V)^* \times (C \cup V)^*$. A *system of word equations* is a finite set of equations of the form $\{L_1 = R_1, \ldots, L_n = R_n\}$ for $n > 0$. A *solution* of such a system is a homomorphism $\alpha\colon (C \cup V)^* \rightarrow C^*$ which behaves as an identity on the letters from C and equates all the equations of the system, i.e. $\alpha(L_i) = \alpha(R_i)$ for all $1 \leq i \leq n$. Such a homomorphism is then fully established by a mapping $\alpha\colon V \rightarrow C^*$. A solution is called *non-singular*, if $\alpha(x) \neq \epsilon$ for all $x \in V$. Otherwise we will call it *singular*. We say that the problem for word equations is satisfiable whenever it has a solution.

Makanin in [11] shows that the satisfiability problem for word equations is decidable. This problem is easily seen to be semidecidable. The decidability is established by giving an upper border on the length of the minimal solution. The decidability was later solved in more general setting by Schulz (see [13]) where for each $\alpha(x)$ is given a regular constraint that must be satisfied.

2.1 Notation

In what follows we will use an uniform notation. The set $C = \{a, b, c, \dots\}$ denotes the alphabet of constants and $V = \{x, y, z, \dots\}$ stands for variables (unknowns) with the assumption that $C \cap V = \emptyset$. We will use the same symbol α for the mapping $\alpha\colon V \to C^*$ and its unique extention to the homomorphism $\alpha\colon (C \cup V)^* \to C^*$. Sometimes we write α_x instead of $\alpha(x)$. The symbol for the empty word is written as ϵ and the length of a word w is denoted as $|w|$.

2.2 Pattern Equations

In this paper we focus on a special kind of word equations which we call *pattern equations.*

Definition 1. *A* pattern equation system *is the set* $\{X_1 = A_1, \dots, X_n = A_n\}$ *where* $X_i \in V^*$ *and* $A_i \in C^*$ *for all* $1 \leq i \leq n$*. The* solution *(both singular and non-singular) of the pattern equation system is defined as in the general case.*

Two natural decidability problems (Pattern-Equation and Non-Singular-Pattern-Equation problem) appear in this context. Given a pattern equation system $\{X_1 = A_1, \dots, X_n = A_n\}$ as an instance of the Pattern-Equation problem, the task is to decide whether this system has a solution. If we require the solution to be non-singular we call it the Non-Singular-Pattern-Equation problem. We give an example of a pattern equation system and demonstrate its solutions.

Example 1. Let us have the following system where $C = \{a, b\}$, $V = \{x, y, z\}$ and the pattern equations are $\{xyxy = abbabb,\quad yzy = bbbabbb\}$. A singular solution exists $\alpha(x) = abb,\quad \alpha(y) = \epsilon,\quad \alpha(z) = bbbabbb$, however, there is also a non-singular solution $\beta(x) = a,\quad \beta(y) = bb,\quad \beta(z) = bab$. There is no reason for having just one solution, which is demonstrated also by our example since $\gamma(x) = ab,\quad \gamma(y) = b,\quad \gamma(z) = bbabb$ is another non-singular solution.

3 NP-Completeness of the Pattern-Equation Problem

In this section we show that the Pattern-Equation problem is NP-complete. First observe that the problem is in NP since any solution is linearly bounded in length w.r.t. the pattern equation system. On the other hand to prove that Pattern-Equation problem is NP-hard we reduce the Tripartite-Matching problem to it. This proof has been independently done in more general setting also by Robson and Diekert [3] using the reduction from 3-SAT.

Suppose we have three sets B, G and H (boys, girls and homes) each containing exactly n elements for a natural number n. Let $T \subseteq B \times G \times H$. The Tripartite-Matching problem is to find a subset $S \subseteq T$ of n elements such that $\{b \in B \mid \exists g \in G, \exists h \in H \colon (b,g,h) \in S\} = B$, $\{g \in G \mid \exists b \in B, \exists h \in H \colon (b,g,h) \in S\} = G$ and $\{h \in H \mid \exists b \in B, \exists g \in G \colon (b,g,h) \in S\} = H$. That is: each boy is matched to a different girl and they have their own home. The Tripartite-Matching problem is known to be NP-complete (see e.g. [12]) and we show a polynomial reduction from it to the Pattern-Equation problem.

Theorem 1. *The* Pattern-Equation *problem is NP-complete.*

Proof. Suppose we have $T \subseteq B \times G \times H$ an instance of the Tripartite-Matching problem where $B = \{b_1, \ldots, b_n\}$, $G = \{g_1, \ldots, g_n\}$ and $H = \{h_1, \ldots, h_n\}$. We will find an instance of Pattern-Equation problem which is satisfiable if and only if the Tripartite-Matching problem has a solution. Let us suppose that $T = \{T_1, \ldots, T_k\}$ and we introduce a new variable t_i for each T_i where $1 \leq i \leq k$. Let us define

$$\phi_B \equiv \wedge_{i=1}^{n} \vee \{t_j \mid \exists g \in G, \exists h \in H \colon (b_i, g, h) = T_j\},$$

$$\phi_G \equiv \wedge_{i=1}^{n} \vee \{t_j \mid \exists b \in B, \exists h \in H \colon (b, g_i, h) = T_j\},$$

$$\phi_H \equiv \wedge_{i=1}^{n} \vee \{t_j \mid \exists b \in B, \exists g \in G \colon (b, g, h_i) = T_j\}.$$

Let us consider the formula $\phi \equiv \phi_B \wedge \phi_G \wedge \phi_H$. We can see that the Tripartite-Matching problem has a solution if and only if there exists a valuation that satisfies the formula ϕ such that it assigns value true to the exactly one variable in each clause. Observe that ϕ is of the form $\phi \equiv C_1 \wedge C_2 \wedge \cdots \wedge C_{3n}$ and assume that there is an empty clause in the conjunction (the formula is not satisfiable). Then we assign it the pattern equation system $\{x = a,\ x = b\}$ (this system certainly does not have any solution). In the other case we may suppose that

$$C_i \equiv t_{i,1} \vee t_{i,2} \vee \cdots \vee t_{i,j_i},$$

where $1 \leq j_i$ for all $1 \leq i \leq 3n$. Then we assign it the following pattern equation system $\mathcal{P}$:

$$\begin{array}{llll} \{ & t_{1,1} \ \ldots \ t_{1,j_1} & = a, & \\ & t_{2,1} \ \ldots \ t_{2,j_2} & = a, & \\ & \ \vdots \qquad\quad \vdots & \ \vdots & \\ & t_{3n,1} \ldots \ t_{3n,j_{3n}} & = a & \} \end{array}$$

The situation when the variable $t_{i,j}$ is true corresponds to $\alpha(t_{i,j}) = a$ and if $t_{i,j}$ is false it corresponds to $\alpha(t_{i,j}) = \epsilon$. It is straightforward that ϕ has a valuation that assigns value true to the exactly one variable in each clause if and only if $\mathcal{P}$ is satisfiable. □

Using a similar proof technique (where the value true is represented by $\alpha(t_{i,j}) = aa$ and false by $\alpha(t_{i,j}) = a$) we can also easily see the validity of the following theorem.

Theorem 2. NON-SINGULAR-PATTERN-EQUATION *problem is NP-complete.*

Remark 1. Observe that for the NP-completeness it is sufficient to fix the constant alphabet just to one letter.

4 Elimination of Conjunction and Disjunction

In general case we may construct for an arbitrary system of word equations a single equation preserving solutions. For example Diekert in [7] used the following construction: the system $\{L_1 = R_1, \ldots, L_n = R_n\}$ and the equation

$$L_1 a \ldots L_n a L_1 b \ldots L_n b = R_1 a \ldots R_n a R_1 b \ldots R_n b$$

where a, b are distinct constants, have the same set of solutions. However, this construction is useless for the pattern equations. We show the way how to eliminate conjunction of pattern equations in the following theorem.

Theorem 3. *The set of solutions of a pattern equation system* $\{X = A,\ Y = B\}$ *is identical with the set of solutions of the pattern equation* $X^n Y^m = A^n B^m$ *where* $n = \max\{|A|, |B|\} + 3$ *and* $m = n + 1$.

Proof. It is evident that each solution of the system $\{X = A,\ Y = B\}$ is also a solution of $X^n Y^m = A^n B^m$. We need the following lemma to prove the opposite.

Lemma 1 ([8]). *Let* $A, B \in C^*$, $d = \gcd(|A|, |B|)$. *If two powers* A^p *and* B^q *of* A *and* B *have a common prefix of length at least equal to* $|A| + |B| - d$, *then* A *and* B *are powers of the same word.*

Let α be a solution of the equation $X^n Y^m = A^n B^m$. We will show that $|\alpha(X)| = |A|$. In such a case $\alpha(X) = A$, $\alpha(Y) = B$ and α is a solution of the system.

First suppose $|\alpha(X)| > |A|$. Then A^n and $\alpha(X)^n$ have a common prefix of length $n \cdot |A|$ and for $|A| > 0$ we get

$$n\,|A| = 2\,|A| + (n-2)\,|A| \geq 2\,|A| + \frac{n+1}{n}(n-3) \geq$$

$$2\,|A| + \frac{m}{n}\,|B| = |A| + \frac{n\,|A| + m\,|B|}{n} \geq |A| + |\alpha(X)|\,.$$

By Lemma 1 we know that $A = D^k$ and $\alpha(X) = D^i$ where $k, i \in \mathbb{N}$, $k < i$, $D \in C^*$ and D is primitive (it means that if $D = E^p$ then $p = 1$). If $|A| = 0$ then trivially $k = 0$ and D is a primitive root of $\alpha(X)$. Hence $D^{(i-k)n}\alpha(Y)^m = B^m$

and by the Lemma 1 (common prefix of length $(i-k)n\,|D| \geq n\,|D| = |D| + (n-1)\,|D| \geq |D| + |B|$) we have that B and D must be the powers of the same word. Since D is primitive we may write $B = D^l$ where $l \in \mathbb{N}_0$. Finally $\alpha(Y) = D^j$ again by Lemma 1.

If $|\alpha(X)| < |A|$ we have $|\alpha(Y)| > |B|$ and we can similarly deduce the same equalities $\alpha(X) = D^i$, $\alpha(Y) = D^j$, $A = D^k$, $B = D^l$.

Now we solve an equation $ni+mj = nk+ml$ in non-negative integer numbers. The proof is complete if we show that this equation has only one solution, namely $i = k$, $j = l$. We recall that $k, l < n, m$ and $m = n+1$. If i, j are such that $ni + mj = nk + ml$ then $i \equiv k \pmod{m}$ and if $i < k+m$ then $i = k$ and $j = l$. Suppose $i \geq k+m$. This implies that $ni+mj \geq ni \geq nk+nm > nk+ml$, which is a contradiction. □

Remark 2. The above construction is unfortunately quadratic in space. One can ask whether n and m in the Theorem 3 need to be greater than $|A|$ and $|B|$? The answer is positive and no improvements can be done. If we want to transform the system $\{x = c^k,\ y = c^l\}$ into a single equation (w.l.o.g. suppose that the equation is of the form $x^n y^m = c^p$) then in the case $l > n$ we have $p = nk+ml = n(k+m)+m(l-n)$ and an α defined by $\alpha(x) = c^{k+m}$, $\alpha(y) = c^{l-n}$ is a solution of the equation $x^n y^m = c^p$ whereas α is not a solution of $\{x = c^k,\ y = c^l\}$.

Remark 3. It is easy to see that the proof of the Theorem 3 is correct for an arbitrary n greater than $\max\{|A|, |B|\}+3$ and $m = n+1$.

The Remark 2 shows that the construction in Theorem 3 can not be improved and moreover every construction preserving the alphabet of variables and constants requires a quadratic space extention. For a system of n equations where each one is bounded by the maximal length k we can repeatedly use the Theorem 3 pairwise and thus achieve the $\mathcal{O}(k^n)$ bound for the size of the resulted equation. On the other side the problem of conjunction elimination can be solved easily with extention of the sets C and V. This is much more suitable for practical purposes since the following construction is linear in space w.r.t. inputed pattern equation system.

Lemma 2. *Let $c \notin C$ be a new constant and $z \notin V$ be a new variable. Then the pattern equation system $\{X = A,\ Y = B\}$ over C, V and the pattern equation*

$$z(zXzY)^2 = c(cAcB)^2 \tag{1}$$

over $C \cup \{c\}$, $V \cup \{z\}$ are equivalent on the set V.

Proof. For every solution α of the system $\{X = A,\ Y = B\}$ we can easily construct a solution α' of the equation (1) such that $\alpha'|_V = \alpha$ and $\alpha'(z) = c$.

Now let α be a solution of the equation (1), i.e.

$$\alpha_z(\alpha_z\alpha_X\alpha_z\alpha_Y)^2 = c(cAcB)^2.$$

If $|\alpha_z| > 1$ then α_z has the prefix c^2 and on the lefthand side of the equality we have at least ten occurrences of c, however, on the righthand side of the equality only five. If $\alpha_z = \epsilon$ then $(\alpha_X \alpha_Y)^2 = c(cAcB)^2$ and the word on the lefthand side of the equality has even length while the word on the righthand side of the equality has odd length. For that reasons $\alpha(z) = c$, hence $\alpha(X) = A$ and $\alpha(Y) = B$. This means that $\alpha|_V$ is a solution of $\{X = A,\ Y = B\}$. □

Remark 4. If we want to find a single equation equivalent to the pattern equation system $\{X_1 = A_1, \ldots, X_n = A_n\}$ we can repeatedly eliminate it pairwise. However, this construction exceeds the linear growth in size. The elimination can be done much better by

$$z(zX_1zX_2z\ldots zX_n)^2 = c(cA_1cA_2c\ldots cA_n)^2$$

and the proof is similar to the previous one.

For disjunction we cannot expect theorems analogical to those we have given for conjunction. For example the disjunction pattern equation system $\{x = c,\ x = c^2\}$ cannot be replaced by a single equation over $\{c\}$, $\{x\}$.

Definition 2. *We say that a homomorphism α is a* solution of the disjunction pattern equation system $\{X_1 = A_1, \ldots, X_n = A_n\}$ *if and only if $\alpha(X_i) = A_i$ for some i, $1 \leq i \leq n$.*

Lemma 3. *Let $c \notin C$ be a new constant and $z_1, z_2, z_3 \notin V$ be new variables. Then the disjunction pattern equation system $\{X = A,\ X = B\}$ over C, V and the pattern equation*

$$z_1 X^{10} z_1^2 z_2^{10} z_3^2 = cA^{10}c^2B^{10}(cA^{10}c^2)^2 \tag{2}$$

over $C \cup \{c\}$, $V \cup \{z_1, z_2, z_3\}$ are equivalent on the set V.

Proof. It is easy to see that if $\alpha(X) = A$ then α' defined as $\alpha'|_V = \alpha$, $\alpha'(z_1) = c$, $\alpha'(z_2) = B$ and $\alpha'(z_3) = cA^{10}c^2$ is a solution of the equation (2). If $\alpha(X) = B$ then $\alpha'(z_1) = cA^{10}c^2$, $\alpha'(z_2) = \alpha'(z_3) = \epsilon$ is also a solution.

Let α be a solution of the equation (2), i.e.

$$\alpha_{z_1}\alpha_X^{10}\alpha_{z_1}^2\alpha_{z_2}^{10}\alpha_{z_3}^2 = cA^{10}c^2B^{10}(cA^{10}c^2)^2.$$

The number of occurrences of c on the righthand side of the equation implies that $\alpha(X)$ and $\alpha(z_2)$ do not contain any c. Moreover if we denote p (resp. q) the number of occurrences of c in $\alpha(z_1)$ (resp. $\alpha(z_3)$) we get $3p + 2q = 9$. This implies that $p = 1$ or $p = 3$. The first case constrains $\alpha(z_1) = c$ and so $\alpha(X) = A$ and the second one gives $\alpha(z_1) = cA^{10}c^2$, hence $\alpha(X) = B$. □

Corollary 1. *For an arbitrary finite set $\mathcal{S} = \{\alpha_i \colon V \to C^* \mid 1 \leq i \leq n\}$ there is a pattern equation over some C', V' such that the set of all its solutions restricted to V is identical with the given set $\mathcal{S}$.*

Proof. First, for every α_i we construct an equation $X_i = A_i$ with a single solution α_i by using repeatedly the Theorem 3. Moreover, in this construction we can use an universal n and m by Remark 3 and thus achieve the same lefthand sides $X_1 = X_2 = \cdots = X_n$. This yields a disjunction pattern equation system $\{X = A_1, \ldots, X = A_n\}$ which is equivalent to some single pattern equation by repeatedly using the Lemma 3. □

Note that in the case of non-singular solutions we may substitute in the Lemma 2 the equation (1) with $zXzYz = cAcBc$ and in the Lemma 3 the equation (2) with $z_1z_2z_1X^2z_1z_3z_1 = c^3A^2cB^2c^3$. It is easy to verify that all the theorems in this section are then also valid for the case of non-singular solutions.

5 Stuttering Equations

It is sometimes interesting to consider the equations not only over a free monoid but for example in bands. Band is a semigroup where the identity $x^2 = x$ is satisfied. In our context it means that the equalities hold up to multiple occurrences of certain substrings, which we call *stuttering*.

Let us define a binary relation $\rightarrow \subseteq C^* \times C^*$ such that $uvw \rightarrow uvvw$ for $u, v, w \in C^*$ and let $\sim$ be its symmetric and transitive closure, i.e. $\sim := (\rightarrow \cup \rightarrow^{-1})^*$. Then the identity $u = w$ holds in a free band if and only if $u \sim v$ (completeness of equational logic). Suppose we have a stuttering equation system $\{L_1 = R_1, \ldots, L_n = R_n\}$. A *solution* of such a system is a homomorphism $\alpha\colon (C \cup V)^* \rightarrow C^*$ which behaves as an identity on the letters from C and equates all the equations of the system, i.e. $\alpha(L_i) \sim \alpha(R_i)$ for all $1 \leq i \leq n$. We call the system a *stuttering pattern equation system* if the equations are of the form $\{X_1 = A_1, \ldots, X_n = A_n\}$.

The solvability problem for a single stuttering pattern equation $X = A$ is trivial since it si always solvable: $\alpha(x) = A$ for all $x \in V$. On the other hand the system is not always solvable: e.g. $\{x = a,\ x = b\}$ has no solution. This immediately gives that conjunction of stuttering pattern equations cannot be eliminated. In what follows we will exploit the fact that the word problem in bands is decidable (see [4] and its generalization [9]), which is mentioned in the next lemma. Let $w \in C^*$. We define $c(w)$ – the set of all letters that occur in w, $0(w)$ – the longest prefix of w in $card(c(w)) - 1$ letters, $1(w)$ – the longest suffix of w in $card(c(w)) - 1$ letters.

Lemma 4 ([4]). *Let $u, v \in C^*$. Then $u \sim v$ if and only if $c(u) = c(v)$, $0(u) \sim 0(v)$ and $1(u) \sim 1(v)$.*

We introduce the size of the solution α as $size(\alpha) := \max_{x \in V} |\alpha(x)|$. Given a stuttering equation system it is decidable whether the system is satisfiable because of the local finiteness of free idempotent semigroups. Following theorem just gives a precise exponential upper bound on the size of the minimal solution. We write $\alpha \sim \beta$ whenever $\alpha(x) \sim \beta(x)$ for all $x \in V$.

Theorem 4. *Let $\{L_1 = R_1, \ldots, L_n = R_n\}$ be a general stuttering equation system where $card(C) \geq 2$. If the system is satisfiable then there exists a solution α such that $size(\alpha) \leq 2^{card(C)} + 2^{card(C)-2} - 2$.*

Proof. Suppose the system is satisfiable, i.e. there is a solution β. We know that any α, $\alpha \sim \beta$, is also a solution. We will find such an α which is small enough. The proof will be done by induction on k where $k = card(C)$.

$k = 2$: The longest minimal word over a two-letter alphabet is of the length 3.
Induction Step: Suppose the IH holds for k and we show its validity for $k+1$. For each $w := \beta(x)$, $x \in V$, we will find some w' such that $w' \sim w$ and $|w'| \leq 2^{k+1} + 2^{k-1} - 2$. We know that $w \sim 0(w)a_1a_21(w)$ where $\{a_1\} = c(w) - c(0(w))$ and $\{a_2\} = c(w) - c(1(w))$ – see Lemma 4. Since $0(w)$ and $1(w)$ are in k letters, the IH can be applied and we can find some u, v of length less or equal $2^k + 2^{k-2} - 2$ such that $u \sim 0(w)$ and $v \sim 1(w)$. Thus we get that $ua_1a_2v \sim w$ and $|ua_1a_2v| \leq (2^k + 2^{k-2} - 2) + 2 + (2^k + 2^{k-2} - 2) = 2^{k+1} + 2^{k-1} - 2$. □

We can construct for each k, $k \geq 2$, a minimal word w_k which is of the length $2^k + 2^{k-2} - 2$ in the following way. Let $w_2 := a_1a_2a_1$ and $w_{k+1} := w_k a_{k+1} a_k w_k[a_k \mapsto a_{k+1}]$ where $w_k[a_k \mapsto a_{k+1}]$ means a substitution of a_k with a_{k+1} in the word w_k. This shows that the border given by the Theorem 4 is tight.

In general it can be shown that there are stuttering equation systems such that all their minimal solutions are exponentially large w.r.t. number of letters from which it consists. Suppose the following sequence of equations: $z_1 = a_1$ and $z_{i+1} = z_i a_{i+1} z_i$ for a pairwise different sequence of constants $a_1, a_2, \ldots$. There is only one minimal solution α of the system and $|\alpha(z_i)| = 2^i - 1$.

An interesting question is whether a minimal solution of a stuttering *pattern* equation system can be of exponential length, too. In fact it turns out [10] that it is always of polynomial length. This long and technical proof exploits the confluent and terminating word rewriting system for bands by Siekmann and Szabo in [14]. Moreover, the NP-hardness of the satisfiability problem is shown.

Theorem 5 ([10]). *The decision problem whether a stuttering pattern equation system is satisfiable is NP-complete.*

Acknowledegments

The authors would like to thank the anonymous referee for pointing out the paper [3].

References

1. Brim L., Gilbert D. R., Jacquet J.-M., Křetínský M.: A Process Algebra for Synchronous Concurrent Constraint Programming, Proceedings of Fifth International Conference on Algebraic and Logic Programming (ALP '96), LNCS volume **1139** (1996) 165–178, Springer-Verlag. 370

2. Černá I., Klíma O., Srba J.: On the Pattern Equations, Technical Report FIMU-RS-99-01, Faculty of Informatics, Masaryk University Brno (1999). 370
3. Diekert V., Robson J. M.: On Quadratic Word Equations, In Proceedings of 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS '99), LNCS volume **1563** (1999) 217–226, Springer-Verlag. 372, 377
4. Green J. A., Rees D.: On semigroups in which $x^r = x$, Proc. Camb. Phil. Soc. **48** (1952) 35–40. 376, 376
5. Kopeček I.: Automatic Segmentation into Syllable Segments, Proceedings of First International Conference on Language Resources and Evaluation (1998) 1275–1279. 369
6. Kopeček I., Pala K.: Prosody Modelling for Syllable-Based Speech Synthesis, Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing, Cancun (1998) 134–137. 369
7. Lothaire M.: Algebraic Combinatorics on Words, Preliminary version available at http://www-igm.univ-mlv.fr/ berstel/Lothaire/index.html. 373
8. Lothaire, M.: Combinatorics on Words, Volume **17** of Encyclopedia of Mathematics and its Applications (1983) Addison-Wesley. 373
9. Kad'ourek J., Polák L.: On free semigroups satisfying $x^r = x$, Simon Stevin **64**, No. 1 (1990) 3–19. 376
10. Klíma O., Srba J.: Complexity Issues of the Pattern Equations in Idempotent Semigroups, Technical Report FIMU-RS-99-02, Faculty of Informatics, Masaryk University (1999). 377, 377
11. Makanin, G. S.: The Problem of Solvability of Equations in a Free Semigroup, Mat. Sbornik. **103(2)** (1977) 147–236. (In Russian) English translation in: Math. USSR Sbornik **32** (1977) 129–198. 369, 371
12. Papadimitriou, C. H.: Computational Complexity, Addison-Wesley Publishing Company (1994), Reading, Mass. 372
13. Schulz, K. U.: Makanin's Algorithm for Word Equations: Two Improvements and a Generalization, In Schulz, K.–U. (Ed.), Proceedings of Word Equations and Related Topics, 1st International Workshop, IWW-ERT'90, Tübingen, Germany, LNCS volume **572** (1992) 85–150, Berlin-Heidelberg-New York, Springer Verlag. 369, 370, 371
14. Siekmann J., Szabó P.: A Noetherian and Confluent Rewrite System for Idempotent Semigroups, Semigroup Forum **25** (1982). 377

Garbage Collection for Mobile and Replicated Objects*

Pablo Galdámez, Francesc D. Muñoz-Escoí, and José M. Bernabéu-Aubán

Instituto Tecnológico de Informática, Universidad Politècnica de València
Camino de Vera s/n, 46071 València, Spain
{pgaldam,fmunyoz,josep}@iti.upv.es

Abstract. This paper describes the Hidra garbage collection system for non-migratable singleton objects, migratable objects and replicated objects. Garbage detection is based on a distributed reference counting protocol and on the use of forwarders when migration occurs. The algorithm is triggered by object reference operations and is completely asynchronous, enabling techniques such as piggy-backing and batching of messages. Once some object is computed as garbage, the ORB sends an unreferenced notification to its implementation. In the case of replicated objects, the unreferenced notification is asynchronously delivered to every replica implementation. The algorithm is safe and live in the absence of failures. When failures arise a reference count reconstruction protocol reestablishes the garbage detection algorithm invariants.

1 Introduction

Distributed systems with object support are becoming of increasing importance. CORBA [13] is one of the most outstanding efforts to standardize and spread their availability. It provides an object model, a common structure for object request brokers and a collection of services and facilities to allow the development of distributed object oriented applications. However support for object replication and garbage detection has not been completed yet. This support is essential for long lived systems with high availability demands.

In this paper we show some aspects of the Hidra object references together with the garbage collection system which is integrated with them. Hidra [6] is the core of a future object oriented distributed operating system and its central component is a fault tolerant object request broker. The ORB handles references for non-migratable, migratable and replicated objects and provides mechanisms to invoke them reliably [11].

Our object references are structured to allow the adoption of several replication strategies such as active replication [17], passive replication [3] or coordinator-cohort replication [1]. The garbage detection (GD) system is independent of any particular replication model. If an object becomes garbage, the

* This work was partially supported by the CICYT (Comisión Interministerial de Ciencia y Tecnología) under project TIC96-0729.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 379–387, 1999.

ORB will send an unreferenced notification to its implementation. The implementation may be unique in the case of singleton objects or multiple for replicated objects. If an object migrates, space is temporarily maintained at the previous object locations until every client reference has been updated accordingly. When this reconnection process terminates, the ORB also collects this temporary garbage. Once every client has reconnected, the GD works as if the object had not migrated.

Our GD only detects acyclic garbage. Cyclic garbage detection [15] is out of the Hidra GD objectives. The mechanism is live in that it eventually detects the garbage that it may appear. It is safe in that as long as some client reference exists for a given object, the object implementation (the replicas implementations) will not receive an unreferenced notification. It is cheap in space and messages. It is inexpensive in space in that object entry points just maintain a counter in opposition to reference listing garbage detection techniques [10,18]. Messages generated by the GD are asynchronous and no extra foreground messages or system activity is required for it to work. The GD messages can be piggy-backed and batched and no multicast protocols nor transaction mechanisms are required.

The Hidra transport protocol provides reliable message passing using a machine level membership monitor [12]. This protocol ensures that a message send from one node to another either reaches its destination or some of the endpoint nodes are considered to have failed. We assume fail-stop [16] behavior, so crashed nodes do not send messages. Based on the Hidra transport reliability, the GD assumes that messages cannot be lost nor duplicated. This assumption at the GD level is realistic since message unreliability is addressed in Hidra at the message transport level [4].

However messages may arrive disordered to their destinations and nodes may fail. When a node crashes a reference count reconstruction protocol is run to reestablish the GD invariants.

Garbage may accumulate in the system if the GD messages take too much time to be sent. This situation is rare given that messages between nodes in a clustered systems are frequent and the GD messages can be piggy-backed into them. In any case, forcing the GD messages to be sent tightens the liveness conditions.

The rest of the paper is structured as follows. Section 2 briefly outlines the Hidra object reference structuring. Section 3 explains the garbage detection system integrated into the ORB. It also introduces the reference counting reconstruction protocol that makes the system fault tolerant. Finally section 4 compares the system with other environments with built-in garbage detection giving some concluding remarks.

2 Object References Overview

Object support in Hidra is built by two layers: the handlers layer and the xdoors layer. Handlers are similar to the Spring subcontracts [7] and to Solaris-MC handlers [8]. Hidra xdoors are based on Solaris-MC xdoors. With respect to garbage detection, handlers simply count their locally managed references whenever object references are received, duplicated or released. When their counter reaches the zero value, handlers notify their underlying xdoor about this fact. This notification tells the xdoor that there are no locally managed references enabling the xdoor space reclamation.

Xdoors are the Hidra internal object references. They are stored at the kernel level and they hold the object OID and the object locator. The OID in a unique identifier which does not change during the object lifetime. The OID is created and assigned to an object xdoor when it is created. Xdoors are created at the server side the first time they are exported to other nodes, thus OIDs are assigned the first time an object is exported.

2.1 Locators

Besides the OID, xdoors hold the object locator. Locators point to the object implementation location and they are mainly used on object invocations made from client xdoors. The contents and structure of object locators depend on the kind of object being considered. For singleton, non-migratable objects, locators have no extra state since the implementation location is fully identified by the OID. For these objects we say that the OID is the object locator. Locators with this simple structure are called *plain locators* and they point to a single node implementation location.

For singleton migratable objects, locators point to one implementation location. This location may be different from the OID when migration has occurred. It is also possible that different client xdoors have different locators for the same object. This happens when migration occurs and there existed client references pointing to the old location.

Finally, for the case of replicated objects, xdoor locators are formed by a set of plain locators. Each one of these plain locators points to one replica implementation location. Different replication models like passive, active or coordinator-cohort replication may have different locator structure. Its even possible to have different locators structuring based on particular replication model implementations. It is out of the scope of this paper to fully describe the locator structuring possibilities. To explain the GD system it is enough to mention that a set of plain locators form each replicated object locator.

Locators are attached to objects at object creation and are propagated to clients by marshaling the locator at the reference donor site and unmarshaling the locator at the receiver site. Some locator classes may marshal themselves by placing just a plain locator on the marshal stream and other may place the complete set of plain locators.

2.2 The Main Xdoor

Mobile objects (migratable singleton objects or replicated objects) may have at a given instant of time several server xdoors. Among the set of server xdoors for a given object there is always one of them playing a special role in several protocols run at the xdoors layer. This special server xdoor is called the *main xdoor*.

For replicated objects the main xdoor is one of its server xdoors. This special server xdoor breaks the symmetry among them to run some distributed protocols needed in Hidra. Examples of such protocols are administrative tasks on replicated objects like replica addition or replica removal, or the distributed reference counting protocol. The concept of main xdoor allows us to redefine *migration* for the case of replicated objects. We say that *a replicated object migrates if and only if the main xdoor migrates*. Every other reconfiguration made on a replicated object, either voluntarily or forced by node failures is not considered as migration.

2.3 The Main Locator

Plain locators may be inaccurate for migratable and replicated objects since nodes may fail and administrative tasks may be performed over those objects. The most accurate plain locator that any locator holds is the *main locator*. In the absence of migrations this locator always points to the main xdoor. When migration occurs, the main locator may be inaccurate.

For singleton non-migratable objects, the OID is always the main locator. For singleton migratable objects, the unique locator is the main locator which may be inaccurate. Finally for replicated objects, whichever replication model is considered, there always exists the main locator which in the absence of migration and failures points to the main xdoor.

3 Garbage Collection

The primary goal of the Hidra garbage collection system is to ensure the delivery of an unreferenced notification to those objects that have not object references pointing to them. As Hidra supports object replication, the unreferenced notification has to reach every object replica implementation. Other system objectives include collection of temporarily allocated space, asynchrony, efficiency and robustness.

Besides ensuring the unreferenced notification, the system has to collect any temporal entities created during the object lifetime. Thus client xdoors located at nodes that have released their object references will be collected. Also, for migrated objects, the system collects server xdoors placed at the previous object locations.

In order to give a first overview of the protocol, we start describing the distributed reference counting protocol in the absence of migration and node failures. In the rest of the section we will show the required extensions to cope with migration and failures.

Our protocol assumes asynchronous and not necessarily ordered communication. The protocol requires a counter called *refcount* as the only state stored at every xdoor for GD purposes. This counter is initialized to 0 on xdoor creation. Our protocol is a variation of the protocol implemented in Solaris-MC [8] and it works as follows:

1. Each time an object reference is sent out of a node the xdoor (server or client) adds one to its *refcount*.
2. When a node receives an object reference that results in the creation of a client xdoor, (there was no xdoor for that object in the node), it tests whether the sender is the node pointed by the main locator contained in the reference.
   (a) If the reference donor is the node of the main locator, nothing else has to be done.
   (b) In other case, the reference receiver adds one to its *refcount* and sends a *INC* message to the main locator. The message includes the address of the node that sent the reference in the first place.
3. When a domain receives an object reference it already had (there exists an xdoor for that object), it sends a *DEC* message to the reference sender.
4. When a node releases all of its object references for a given object, that is, when the handler notifies this fact to the xdoor, the xdoor tests its *refcount* value.
   (a) If the *refcount* value is zero, then the following steps are given:
      i. If the xdoor is not the main xdoor, a *DEC* message is sent to the main locator.
      ii. If the xdoor is a client xdoor, the xdoor is collected as garbage, leaving this node without any state referring to the object.
      iii. If the xdoor is the main xdoor, then the object is unreferenced. In this case an *UNREF* message is sent to each one of the object server xdoors[1].
   (b) If the *refcount* value is greater than zero, the xdoor is marked as *collectable* and nothing else is done. This node has no object references but the xdoor cannot be collected yet since the GD invariants rely on its counter.
5. When a xdoor receives a *DEC* message, it subtracts one from its *refcount*. If the *refcount* reaches zero and the xdoor is marked as *collectable*, the step 4a is executed.
6. Whenever a node receives an object reference the *collectable* flag is removed from the receiving xdoor.
7. When a xdoor receives an *INC* message, it adds one to its *refcount* and sends two *DEC* messages. One to the sender of the *INC* message and the other to the address contained in the message. Note that *INC* messages are sent to main locators which always point to server xdoors. We will see later how these messages are managed by migrated objects.
8. When an xdoor receives an *UNREF* message, an unreferenced notification is delivered to its implementation and the xdoor is collected.

[1] In this paper we assume that main xdoors hold a plain locator for every replica implementation or at least that the have a means to obtain all of them.

A formal demonstration of the algorithm safety and liveness conditions are beyond the scope of this paper. However these properties are easily observed focusing our attention on the *refcount* values, messages in transit and fair computations.

3.1 Garbage Collection for Migratable Objects

When an object migrates, there could appear xdoors with invalid main locators. So the previous algorithm would fail in this case. Migration may occur as the result of administrative tasks or in the case of replicated objects when the main xdoor node fails. To simplify the exposition we will assume that the migration protocol performs the following tasks atomically in respect to any other object reference activity:

1. Move the main xdoor role from the current main xdoor to the new one.
2. Put a main locator at the old main xdoor location pointing to the new main xdoor location.
3. Mark the old main xdoor as migrated.
4. Refcount values at the old main xdoor and at the new main xdoor are updated to reflect the new situation (Both could change in one unit).

In several situations a *MIGRATED* message can be generated. For instance, if an old main xdoor receives an *INC* message it works as if it was the main xdoor but it also sends a *MIGRATED* message to both the message sender and the destination included in the message.

When an xdoor receives a *MIGRATED* message and the contained main locator is not the same as the xdoor main locator, the xdoor updates its main locator, increases its *refcount* value and it sends an *INC* message to the received main locator. This *INC* message includes the address of the old main locator. *MIGRATED* messages that arrive to already collected xdoors are simply discarded.

3.2 Reference Counting Fault Tolerance

On node crashes, the Hidra membership monitor [12] allows the execution of distributed reconfiguration steps executed by the surviving nodes in a synchronous lock step way. One of the first steps given in Hidra is the execution of a reference counting reconstruction protocol whose goal is to reestablish the GD invariants. These invariants have been possibly violated due to in-transit message losses while the crash occurred.

The reconstruction protocol is quite simple but expensive (quadratic message cost and two rounds). During the first round, main xdoor locations publish themselves at a known location and during the second round each node contacts this known location to rebuild the counting. The final goal of this protocol is to reach an state in which every main xdoor has the exact count of client xdoors that exist in the cluster. Further details of this protocol can be found in [5].

4 Related Work and Conclusions

In this paper we have described a distributed garbage collector for non-migratable, migratable and replicated objects. Garbage collection is addressed in several works. They are surveyed in [19] for centralized systems and in [14] focusing on distribution. Some of these works have similarities to ours.

Garbage collection for migratable objects is addressed in [18] using a form of reference listing called the stub-scion pairs. This system tolerates several message failures and copes with node failures. Our GD just considers message disordering and node crashes since other kind of message failures are treated at the transport level. We strongly believe that our approach is more usable since message losses and duplicates cannot appear on object invocations nor reference passing. On the other hand, they suggest a range of possibilities to cope with node failures that break the stub-scion pair chains but it seems that no one of them is finally implemented.

Garbage collection for network objects [2] is similar to ours. They store at the server side a list of entry points for each external object reference pointing to the object. This scheme results in high spatial needs and makes it difficult to support object migration. Fault tolerance is achieved sending *ping* messages to the clients in order to test their availability. Our scheme is less space demanding and tolerates failures by using a failure suspector which serves for other system purposes and not only for garbage collection.

Our reference counting protocol is a variant of the protocol presented in [8] which in turn share many similarities with the network objects one. In Solaris-MC a *reference registry* is done by the reference receiver when the reference sender is not the implementation holder. In that case the reference receiver, increments its *refcount* counter to prevent xdoor reclamation and sends an *INC* message to the implementation holder. When the server receives an *INC* message, it replies with an *ACK* to the reference receiver which in turn sends a *DEC* message to the node that gave the reference in the first place. This scheme has the disadvantage that reference receivers have to "remember" *INC* messages sent to implementation holders to wait for their *ACK* messages. Our approaches makes the protocol more asynchronous since no node has to remember any reference passing event.

No one of the three works analyzed provide support for object replication.

Our system considers replicated objects as first level objects. A number of systems exist (see [9] as an example) where replicated objects are built grouping a range of non-replicated references. In those systems we say that replicated objects are second-level objects since there is little internal support for them. In Hidra we consider replicated objects as first level objects which receive the same support as non-replicated ones including garbage collection. If garbage collection is required for systems with second level replicated objects, it would result in a more expensive solution than ours. Our approach is cheaper since space and messages generated for garbage collection is similar to the required for non-replicated reference management and is not proportional to the number of replicas of each replicated object.

References

1. K. P. Birman, T. Joseph, T. Raeuchle, and A. El Abbadi. Implementing fault-tolerant distributed objects. *IEEE Trans. on SW Eng.* (1985) 11(6):502–508. 379
2. Andrew Birrell, David Evers, Greg Nelson, Susan Owicki, and Edward Wobber. Distributed garbage collection for network objects. TR-116, DEC Research (1993). 385
3. N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. The primary-backup approach. *Distributed Systems (2^{nd} ed.)*, Addison-Wesley, (1993) 199–216. 379
4. P. Galdámez, F. D. Muñoz, and J. M. Bernabéu-Aubán. HIDRA: Architecture and high availability support. TR DSIC-II/14/97, U. Politècnica de València, Spain, (1997). 380
5. P. Galdámez, F. D. Muñoz, and J. M. Bernabéu-Aubán. Garbage notification in a kernel based ORB. Proc. of the VI Jornadas de Concurrencia, Gandia, Spain, (1999). 384
6. P. Galdámez, F. D. Muñoz-Escoí, and J. M. Bernabéu-Aubán. High availability support in CORBA environments. In F. Plášil and K. G. Jeffery, (Eds.), *24^{th} SOFSEM, Milovy, Czech Republic*, Vol. 1338 LNCS, Springer Verlag (1997) 407–414. 379
7. Graham Hamilton, Michael L. Powell, and James J. Mitchell. Subcontract: A flexible base for distributed programming. In Barbara Liskov, editor, *Proceedings of the 14^{th} Symposium on Operating Systems Principles*, New York, NY, USA, (1993) 69–79. 380
8. Y. A. Khalidi, J. M. Bernabéu, V. Matena, K. Shirriff, and M. Thadani. Solaris MC: A multi computer OS. USENIX Conference Proceedings 1996, Berkeley, CA, USA, (1996) 191–203. 381, 383, 385
9. S. Maffeis. *Run-Time Support for Object-Oriented Distributed Programming.* PhD thesis, Dept. of Comp. Sc., Univ. of Zurich (1995). 385
10. U. Maheshwari and B. H. Liskov. Fault-tolerant distributed garbage collection in a client-server object-oriented database. In *Parallel and Distributed Information Systems (PDIS '94)*, IEEE Computer Society Press. (1994) 239–248. 380
11. F. D. Muñoz-Escoí, P. Galdámez, and J. M. Bernabéu-Aubán. ROI: An invocation mechanism for replicated objects. In *Proc. of the 17^{th} IEEE SRDS* (1998) 29–35. 379
12. F. D. Muñoz-Escoí, V. Matena, J. M. Bernabéu-Aubán, and P. Galdámez. A membership protocol for multi-computer clusters. TR DSIC-II/20/97, Univ. Politècnica de València, Spain (1997). 380, 384
13. OMG. *The Common Object Request Broker: Architecture and Specification.* Revision 2.2. Object Management Group (1998). 379
14. David Plainfossé and Marc Shapiro. A survey of distributed garbage collection techniques. *Proceedings of International Workshop on Memory Management*, Vol. 986 of LNCS, Springer-Verlag, (1995). 385
15. Helena C. C. D. Rodrigues and Richard E. Jones. A cyclic distributed garbage collector for Network Objects. *WDAG'96*, Vol. 1151 of LNCS, Springer-Verlag, (1996). 380
16. R. D. Schlichting and F. B. Schneider. Fail-stop processors: An approach to designing fault-tolerant systems. *ACM Trans. on Computer Sys.*, 1(3) (1983). 380
17. F. B. Schneider. Replication management using the state-machine approach. *Distributed Systems (2^{nd} ed.)*, Addison-Wesley, Wokingham, UK, (1993) 166–197. 379

18. Marc Shapiro, Peter Dickman, and David Plainfossé. Robust distributed references and acyclic garbage collection. *Proc. of the 11th Annual Symposium on Principles of Distributed Computing* (1992) 135–146. 380, 385

19. Paul R. Wilson. Uniprocessor garbage collection techniques. *Proc. of International Workshop on Memory Management*, Vol. 637 of LNCS, Springer (1992) 16–18. 385

Randomized Gossiping by Packets in Faulty Networks*

Anna Gambin and Adam Malinowski

Institute of Informatics, Warsaw University
Banacha 2, 02–097 Warszawa, Poland
{aniag,amal}@mimuw.edu.pl

Abstract. Every node of a network has a value which should be made known to all the other nodes. Some *arbitrarily placed* nodes and links may be faulty, however for each pair of fault-free nodes the number of fault-free paths of length 2 that connect them is linear in the size of the network. Faults are of crash type: faulty links do not transmit and faulty nodes neither send nor receive messages. In a unit of time a node can send or receive at most one packet containing the values of at most b nodes. For every $1 \leq b \leq n$ we present an asymptotically time-optimal randomized algorithm to exchange values between all the fault-free nodes with the probability exceeding $1 - n^{-\varepsilon}$, where n is the number of nodes and $\varepsilon > 0$ is an arbitrary constant.

1 Introduction

Gossiping is one of the basic tasks in network communication. Each node of a communication network has a piece of information which needs to be passed to all the other nodes by exchanging messages along the links of the network. There are several extensive surveys on this topic (and closely related broadcasting), see e.g. [5,7].

As networks grow in size, the risk of their components failure rapidly increases, which implies the need to design algorithms able to accomplish given communication task in spite of these faults. The vast literature on fault-tolerant broadcasting and gossiping has been surveyed in [9].

Two models of fault distribution are the most commonly used in the literature: the *bounded* model in which an upper bound on the number of *arbitrarily placed* faulty components is assumed, and the *probabilistic* model in which faults are assumed to occur randomly and independently of each other. Gossiping in the presence of random crash faults was considered in [3] and the related problem of Byzantine Agreement in the presence of random byzantine faults was discussed in [8]: asymptotically time-optimal deterministic algorithms for these problems were developed, working correctly with high probability. This paper adopts a dual approach: we consider *worst case* placement of faults and present a *randomized* solution.

[*] This work was partially supported by the KBN grant 8 T11C 039 15.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 387–394, 1999.

The paper is organized as follows. In Sect. 2 we describe our communication and fault model and give some probabilistic facts to be used in the sequel. Section 3 contains a description of our randomized gossiping algorithm, while in Sect. 4 we prove that it works correctly with high probability. Section 5 contains conclusions and an open problem.

2 Model Description and Preliminaries

The underlying graph of communication network is a directed n-node clique. Communication is performed in synchronous steps. During a single step each node can perform some internal computations and then send or receive a message from at most one other node. SEND and RECEIVE parts of a single step are separated, so a node can both send and receive a message in one step. If more than one node sends a message to node v at the same step then none of them succeeds and v receives nothing.

Some nodes and links of the network may be faulty. The faults are of crash type: a faulty link does not transmit and a faulty node neither sends nor receives messages. The distribution of faulty components is arbitrary, but for any pair of nodes u, v the number of fault-free nodes w such that both links uw and wv are fault-free is at least pn, for some constant $p > 0$. This condition holds if e.g. the total number of faulty components is bounded by $(1-p)n$ or for each node the number of its faulty neighbors and faulty links adjacent to it does not exceed $(1-p)n/2$. We assume the faults to be *static*, i.e. their location is determined prior to the execution of the algorithm. We shall assume that the faults are revealed at the very beginning of the communication process, although our gossiping procedure can be easily adapted to the case when the components, which were predetermined as faulty, can actually fail in an arbitrary moment of the algorithm execution.

Because the algorithm considered in this paper is randomized, a node receiving some value does not *a priori* know whose value it is, so the messages must contain labels of nodes attached to values. Our algorithm is parametrized by the size b of a message: a message of size b, for $1 \leq b \leq n$, can contain at most b pairs $\langle \text{label}, \text{value} \rangle$.

In our probabilistic considerations we will use the following lemma known as Chernoff bound [6]:

Lemma 1. *Let X be the number of successes in a series of b Bernoulli trials with success probability q. Then, for any constant δ with $0 < \delta < 1$, the following inequality holds:*

$$Pr(X \leq (1-\delta)qb) \leq e^{-\delta^2 qb/2}.$$

□

A similar bound also holds if we assume only *limited independence* of a series of trials. Formally, a family A of random variables is called *k-independent* if the variables in any subset $B \subset A$ of size at most k are mutually independent.

The following fact is proved in [10]:

Lemma 2. *Let $X_1, \ldots, X_n$ be binary random variables with $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$. Then for any $0 < \delta \le 1$, if the X_i's are $\lfloor \delta^2 \mu e^{-1/3} \rfloor$-independent then the inequality $Pr(|X - \mu| \ge \delta\mu) \le e^{-\delta^2 \mu/3}$ holds.* □

3 Randomized Gossiping Algorithm

An elementary communication procedure SEND(v, w, M) used in the formal description of our algorithm consists of an attempt of sending message M containing at most b pairs $\langle \text{label}, \text{value} \rangle$ from node v to w. If the size of message M exceeds b then we apply procedure TRANSMIT(v, w, M) which cuts M into chunks of size at most b and SENDs them sequentially; the receiver can concatenate them in order to obtain M:

```
**procedure** TRANSMIT$(v, w, M)$
cut $M$ into chunks $M_1, \ldots, M_{\lceil |M|/b \rceil}$, each of size at most $b$;
**for** $i \leftarrow 1$ **to** $\lceil |M|/b \rceil$ **do**
    SEND$(v, w, M_i)$
```

Procedure TRANSMIT takes $\mathcal{O}(\lceil |M|/b \rceil)$ steps.

Let $k = \max(c_0 \log n, b)$, where b is the maximum size of a message and c_0 is a constant to be specified later. Without loss of generality assume that n is a power of a prime number, so there exists a field $\mathbb{F}$ of size n, and $k \mid n$. We will denote the set of all nodes by V and identify the labels of nodes with elements of $\mathbb{F}$. We also fix some bijection $f \colon \mathbb{F} \to \{0, \ldots, n-1\}$.

The algorithm is divided into three stages. The aim of the first stage is to partition the set of nodes into n/k groups $G_0, \ldots, G_{n/k-1}$ in such a way that with high probability the following holds:

P1: each fault-free node is able to correctly determine the group each node belongs to,

P2: each group contains at most $2k$ nodes,

P3: each group contains at least $pk/2$ fault-free nodes.

This is achieved in the following way: each node v chooses at random the coefficients $a_0, \ldots, a_{k-1}$ of a polynomial $h_v(x) = \sum_{i=0}^{k-1} a_i x^i \in \mathbb{F}[x]$ and sets $l_v = \text{label}(v)$. Now the following operations are repeated $c_1 \log n$ times, where c_1 is a suitable constant: each node v attempts to send a packet (l_v, h_v) containing an auxiliary label and the description of a polynomial to a random neighbor. If a node w receives such a packet and the label l_v is smaller than its own auxiliary label l_w then it replaces its polynomial h_w by h_v and the label l_w by l_v. It will be shown in Sect. 4 that after the first stage each fault-free node holds the same polynomial h with high probability if constant c_1 is sufficiently large. Formally:

```
procedure STAGE-1
for $v \in V$ in parallel do
    $v$ chooses $h_v = \langle a_0, \dots, a_{k-1} \rangle \in F^k$ uniformly at random
    $l_v \leftarrow \mathrm{label}(v)$
repeat $c_1 \log n$ times
    for $v \in V$ in parallel do
        $v$ chooses $w \in V$ uniformly at random
        TRANSMIT$(v, w, \langle l_v, h_v \rangle)$
        if $v$ has received message $\langle l_u, h_u \rangle$ and $f(l_u) < f(l_v)$ then
            $\langle l_v, h_v \rangle \leftarrow \langle l_u, h_u \rangle$
```

STAGE-1 takes time $\mathcal{O}(\frac{k}{b} \log n)$.

Partitioning nodes into groups is performed depending on the values that the polynomial h takes on the labels of nodes. Specifically, a node with the label x belongs to the group $G_{f(h(x)) \operatorname{div} k}$, where div denotes integer division. We will show in Sect. 4 that this method of partitioning nodes into groups fulfills conditions P1–P3.

The second stage is executed under the assumption that all the fault-free nodes hold the same polynomial h. This stage consists of performing gossiping in groups in $c_2 \log n$ steps, for some constant c_2. During each step every node chooses uniformly at random some other node w in its group and an intermediary node $u \in V$ and attempts to send all values of nodes of the group it currently knows to u which in turn passes the information to w. It will be shown in Sect. 4 that after the second stage each fault-free node knows the values of all the fault-free nodes in its group with high probability. In the following pseudocode $\mathrm{val}_v(G_i)$ denotes the set of all pairs $\langle \mathrm{label}, \mathrm{value} \rangle$ of nodes of G_i currently known by v:

```
procedure STAGE-2
repeat $c_2 \log n$ times
    for $v \in V$ in parallel do
        let $G_i$ be the group $v$ belongs to (i.e. $i = f(h(\mathrm{label}(v))) \operatorname{div} k$)
        $v$ chooses $w \in G_i$ uniformly at random
        $v$ chooses $u \in V$ uniformly at random
        TRANSMIT$(v, u, \langle \mathrm{label}(w), \mathrm{val}_v(G_i) \rangle)$
        if $v$ has received message $\langle \mathrm{label}(x), M_y \rangle$ then
            TRANSMIT$(v, x, M_y)$
        if $v$ has received message $M_z$ then
            $\mathrm{val}_v(G_i) \leftarrow \mathrm{val}_v(G_i) \cup M_z$
```

STAGE-2 takes time $\mathcal{O}(\frac{k}{b} \log n)$.

The third stage is executed under the assumption that all the fault-free nodes know the values of all the fault-free nodes in their groups. It is divided into two substages. First each node broadcasts the set of values of its own group to randomly selected nodes for $c_3 n/k$ steps, where c_3 is a suitable constant. After this substage the values of each group are known by a linear number of nodes with high probability. In the second substage every node that does not know the values of nodes of some group sends a query containing its own label and the number of this group to a randomly selected node. If a node receives a query and has the required information, then it answers by sending a message. This process is repeated $c_4(n/k + \log n)$ times, for some appropriate constant c_4. It will be shown that after completing the third stage all nodes have collected the values of all groups with high probability. In the following pseudocode $\mathrm{val}_v(G_i)$ has the same meaning as above, however now we can omit the subscript because each node knows either everything or nothing about each group.

```
procedure STAGE-3
repeat $c_3 n/k$ times
    for $v \in V$ in parallel do
        let $G_i$ be the group $v$ belongs to
        $v$ chooses $w \in V \setminus G_i$ uniformly at random
        TRANSMIT($v, w, \mathrm{val}(G_i)$)
repeat $c_4(n/k + \log n)$ times
    for $v \in V$ in parallel do
        if $v$ does not know the values of nodes of some group then
            let $G_i$ be any such group
            $v$ chooses $w \in V$ uniformly at random
            TRANSMIT($v, w, \langle v, i \rangle$)
        if $v$ receives message $\langle u, j \rangle$ and $v$ knows the values of nodes of $G_j$ then
            TRANSMIT($v, u, \mathrm{val}(G_j)$)
```

STAGE-3 takes time $\mathcal{O}(n/k + \log n)$, so the total time of the algorithm is $\mathcal{O}(n/b + \log n)$ which is clearly optimal.

4 Probability of Correctness

In this section we show that for every $\varepsilon > 0$ there exist constants c_i, for $i = 0, \ldots, 4$ such that the gossiping algorithm described in the previous section works correctly with the probability exceeding $1 - n^{-\varepsilon}$ for sufficiently large n. Let us fix a constant $\varepsilon > 0$.

First we consider the following simple randomized algorithm to broadcast some message M originally held in a single source node s to all other nodes of the network, studied in [4]. This algorithm was also studied in [1] in the context of PRAMs with memory faults.

```
procedure RB
repeat $t$ times
    for $v \in V$ in parallel do
        if $v$ has already got $M$ then
            $v$ chooses $w \in V$ uniformly at random
            TRANSMIT($v, w, M$)
```

It turns out that in our setting it is enough to take $t = \mathcal{O}(\log n)$:

Lemma 3. *For any positive constants α and p there exists a constant c such that if $t \geq c \log n$ and for each pair of fault-free nodes the number of fault-free paths of length 2 that connect them is at least pn then the algorithm RB successfully propagates the original value to all fault-free nodes in V with the probability exceeding $1 - n^{-\alpha}$.*

Proof. (Outline) The proof is an adaptation of the proof from [1]. We partition the execution of procedure RB into two subphases and analyse them separately.

First we show that if the number X of the informed nodes is less than some constant fraction of n then X increases exponentially in each step with constant positive probability. This implies that X reaches the threshold after $\mathcal{O}(\log n)$ steps with high probability.

If X exceeds the threshold then it can be shown that the number of *uninformed* nodes *decreases* exponentially in each step with constant probability, so it drops down to zero after $\mathcal{O}(\log n)$ steps with high probability. □

The following lemma is an immediate consequence of Lemma 3:

Lemma 4. *There exists a constant c_1 such that after the execution of the procedure STAGE-1 every fault-free node knows the same polynomial h with the probability exceeding $1 - 1/4n^{-\varepsilon}$.* □

Now we prove that the remaining two properties P2 and P3 of partitioning nodes into groups induced by the polynomial h also hold with high probability:

Lemma 5. *There exists a constant c_0 such that after the execution of the procedure STAGE-1 with the probability at least $1 - 1/4n^{-\varepsilon}$ the following holds:*

(a) each group contains at most $2k$ nodes,
(b) each group contains at least $pk/2$ fault-free nodes.

Proof. (a) Fix $0 \leq i_0 < n/k$ and let

$$X_v = \begin{cases} 1 \text{ if } v \in G_{i_0}, \\ 0 \text{ otherwise} \end{cases}$$

for $v \in V$. These random variables are k-independent, for a proof see [2].

Let $X = \sum_{v \in V} X_v$ denote the size of the group G_{i_0}. $E[X] = k$, so taking $\delta = 1$ in Lemma 2 we get $\Pr(X \geq 2k) \leq e^{-k/3}$, which is less than $1/8n^{-(1+\varepsilon)}$ for sufficiently large constant c_0. Multiplying this probability by the number of groups $n/k \leq n$ we conclude that the probability of the event A that the size of *any* group exceeds $2k$ is smaller than $1/8n^{-\varepsilon}$.

(b) The reasoning is similar, but now we take

$$X_v = \begin{cases} 1 \text{ if } v \in G_{i_0} \text{ and } v \text{ is fault-free}, \\ 0 \text{ otherwise} \end{cases}$$

and $\delta = 1/2$. $E[X]$ is now at least pk, so $\Pr(X \leq pk/2) \leq e^{-pk/12}$, which again is less than $1/8n^{-(1+\varepsilon)}$ if c_0 is sufficiently large, so the probability of the event B that *any* group contains less than $pk/2$ fault-free nodes is smaller than $1/8n^{-\varepsilon}$.

The probability of the event $A \cup B$ is less than $1/4n^{-\varepsilon}$, as required. □

We proceed with the analysis of the second stage, under the assumption that the conditions P1, P2 and P3 hold.

Lemma 6. *There exists a constant* c_2 *such that after completing STAGE-2 each fault-free node knows the values of all the fault-free nodes in its group with the probability exceeding* $1 - 1/4n^{-\varepsilon}$.

Proof. Let us focus on distributing the value of some fixed node $v \in G_i$ during the execution of STAGE-2. This can be viewed as performing algorithm RB in G_i. By Lemma 3 it follows that the value of v reaches all fault-free nodes in G_i with the probability exceeding $1 - 1/4n^{-(1+\varepsilon)}$, for sufficiently large constant c_2. The same holds for *every* fault-free $v \in V$ with the probability exceeding $1 - 1/4n^{-\varepsilon}$. □

Now we analyse the third stage, under the assumption that each fault-free node knows the values of all the fault-free nodes in its group.

Lemma 7. *There exist constants* c_3, c_4 *such that after completing STAGE-3 every fault-free node knows the values of all fault-free nodes in* V *with the probability exceeding* $1 - 1/4n^{-\varepsilon}$.

Proof. (Outline) After the first substage the values of each group are known by a linear fraction of the nodes with high probability. Distributing the values of the nodes of some fixed group can be viewed as the following process of placing balls into bins: there are $\frac{c_3 n}{k} \cdot \frac{pk}{2} = \Theta(n)$ balls which are randomly and independently placed in at least pn bins. A ball disappears when either some other node simultanously tries to send a message to the same node or a faulty node or link has been chosen. This happens with the probability at most $(1 - (\frac{n-1}{n})^{n-1})(1-p) \leq 2(1-p)/3$. Lemma 1 implies that the number of nonempty bins exceeds $pn/2$ with high probability if c_3 is sufficiently large.

In order to complete the proof it is enough to show that for each fault-free node the expected number of steps necessary to get the values of the nodes from a single group is constant. Fix a fault-free node $v \in V$ and $0 \leq i_0 < n/k$. Denote by A the set of its fault-free neighbors connected to v via fault-free links. If a linear number of nodes from A knows $\mathrm{val}(G_{i_0})$ then v gets the information in a single iteration with constant probability. Otherwise A contains a subset B such that the size of B is linear and each node $w \in B$ is connected by fault-free links to linear number of nodes knowing $\mathrm{val}(G_{i_0})$, hence v gets the information in two iterations with constant probability. □

Lemmas 4, 5, 6, and 7 together give the following

Theorem 1. *For every* $\varepsilon > 0$ *there exist constants* c_0, c_1, c_2, c_3, c_4 *such that the gossiping algorithm described in Sect. 3 works correctly with the probability exceeding* $1 - n^{-\varepsilon}$. □

5 Conclusions

We have presented an asymptotically time-optimal randomized gossiping algorithm, for any message size, in a network with arbitrarily placed faulty nodes and links. The faults (of crash type) were assumed static, i.e. their location is determined prior to the execution of the algorithm. One might consider a stronger model of *dynamic* faults: the faults are transient but the set of faulty components can change after each step, provided that for each pair of fault-free nodes the number of fault-free paths of length 2 that connect them is linear in the size of the network. Actually, in order to make the gossiping problem well defined in this setting we must assume that all the nodes are fault-free. The randomized broadcasting algorithm from [4] works also with dynamic faults, which implies the existence of an asymptotically optimal gossiping algorithm for linear size messages in this fault model, as well as an $\mathcal{O}(n \log n)$ algorithm for constant size messages. The following question seems to be interesting:

Problem: Does there exist an $o(n \log n)$ randomized gossiping algorithm for dynamic link faults and constant size messages, working correctly with the probability exceeding $1 - n^{-\varepsilon}$, where n is the number of nodes and $\varepsilon > 0$ an arbitrary constant?

References

1. B. S. Chlebus, A. Gambin, P. Indyk, PRAM computations resilient to memory faults, Proceedings of the 2nd Annual European Symposium on Algorithms, LNCS 855 (1994), 401–412. 391, 392
2. M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, R. E. Tarjan, Dynamic perfect hashing: upper and lower bounds, SIAM J. Comput. Vol. 23, No. 4 (1994), 738–761. 392
3. K. Diks, A. Pelc, Efficient gossiping by packets in networks with random faults, SIAM J. on Discr. Math. 9 (1996), 7–18. 387
4. U. Feige, D. Peleg, P. Raghavan, E. Upfal, Randomized broadcast in networks, Random Structures and Algorithms 1 (1990), 447–460. 391, 394
5. P. Fraigniaud, E. Lazard, Methods and problems of communication in usual networks, Disc. Appl. Math. 53 (1994), 79–133. 387
6. T. Hagerup, C. Rub, A guided tour of Chernoff bounds, Inf. Proc. Letters 33 (1989/90), 305–308. 388
7. S.M. Hedetniemi, S.T. Hedetniemi, A.L. Liestman, A survey of gossiping and broadcasting in communication networks, Networks 18 (1988) 319–349. 387
8. A. Malinowski, Efficient Byzantine Agreement in networks with random faults, Parallel Proc. Letters Vol. 7, No. 1 (1997), 69–76. 387
9. A. Pelc, Fault-tolerant broadcasting and gossiping in communication networks, Networks 28 (1996), 143–156. 387
10. J. P. Schmidt, A. Siegel, A. Srinivasan, Chernoff-Hoeffding bounds for applications with limited independence, Proceedings of the ACM/SIAM Symposium on Discrete Algorithms (1993), 331–340. 388

Object-Oriented Specification with the Parallel Multi-Label-Selective λ-calculus*

Carlos Herrero and Javier Oliver

DSIC, Universidad Politècnica de València, Camino de Vera s/n
46022 València, Spain
{cherrero,fjoliver}@dsic.upv.es

Abstract. LCEP is a calculus for modeling concurrent systems. The efforts to use it to represent object-oriented features have been successfully treated in previous papers, but some of the problems are unsolved for the lacks of the initial calculus. We present LCMEP, an extension of LCEP which is able to represent the operational semantics of π-calculus and it supports also the commendable characteristics of λ-calculus. By using the power of the calculus, it is possible to represent object-oriented programming language features. We present an operational semantics for a parallel object-oriented programming language by means of a phrase-by-phrase translation from the language into LCMEP.

Keywords: Concurrency, Extensions of λ-calculus, Object-Orientation, Process Algebra.

1 Introduction

A system is described as an object collection in the object-oriented programming world. Each object is defined as an entity which joins data and procedures over these data. In [3], P. America and J. Rutten considered an object as a *black-box* which contains information, and also can manage it. Data inside of an object are stored in variables. These variables could be modified by assignment statements. The variable manipulation is only possible from inside the object. Therefore, we can assume that a private environment exists when we try to use those variables. There is only one way to allow other objects to access this information: sending messages. There are only two kinds of messages: a remote invocation for execution, and the answer over that call. This is the communication way in languages like POOL [2]. It is possible to find many points of view to represent concurrency and parallelism. This paper follows the concepts and solutions previously treated in [3]. The general idea is to use a sequential language to define the behaviour inside the object. The parallelism appears with the parallel composition of different objects. The communication between objects is possible by means of the remote call of methods (client to server) and the answers (server to client). It is necessary to add new statements to allow the synchronous and the asynchronous communication.

* This work was partially supported by CICYT, TIC 98-0445-C03-01.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 395–403, 1999.

In this paper we pay basically attention on the process perspective to parallelism and communication. Each object is presented as a term of the calculus. The universe of objects is the parallel composition of those terms. There exist many approaches to model object-oriented systems in the literature. We could cite valuable contributions using different calculus and paradigms (π-calculus [10]), Petri nets, Øjeblik/Obliq, ...). However, among all desirable characteristics of object-oriented programming we want to remark the basic behavioural structure: the *communication*. This feature is the most interesting from the concurrency modeling perspective. This basic mechanism and its control is the main idea of our proposal. We present an extension of λ-calculus, LCMEP (*Parallel MultiLabel-Selective λ-calculus*). The calculus expresiveness solves problems added to classical programming in a succesfully way by using communications, concurrency and privacy. In a previous proposal [6] we consider the use of LCEP [9] (Parallel Label-Selective λ-calculus). Nevertheless, the lacks of the calculus avoid the right representation of object generation, identification and privacy. Our proposal is to extend LCEP introducing the *environment* concept. The environment has similar characteristics as the *membrane* of the Chemical Abstract Machine [4]. The basical idea is to perform a suitable tool to model the locality of the communications. We consider two fundamental features: first, the communication structure is not altered by the new extension; second, we have a secure method to communicate, maintaining the privacy in any circumstance. Therefore, we model the communications between objects with no collisions against any clients competing for a server, and vice versa.

Section 2 describes the main characteristics of LCMEP. Section 3 shows the translation function of a parallel object-oriented programming language into LCMEP, as an operational semantics of the language. Finally, there are some concluding remarks. In [7] we present an example of class declaration in the programming language and its equivalence in the calculus.

2 The Parallel Multi-Label-Selective λ-calculus

The Parallel Label-Selective λ-calculus (LCEP) [9] is the calculus originated from a previous proposal by H. Aït-Kaci (the label-selective λ-calculus [1]). This calculus is an extension of λ-calculus in which function arguments are selected by labels. Symbolic and numeric labels can be used to name the communication channels. This has not been possible in other proposals and it allows for the use of currying and the labeled specification of parameters. It makes the ordering of the actual parameters in a function call independent of the presentation order of the formal parameters in the definition. Our proposal is an extension of this calculus into the Parallel Multi-Label-Selective λ-calculus (LCMEP) by considering a channel as a simple label or a pair of labels. Let $V = \{x, y, z, \dots\}$ be a set of variables, and $C = \{a, b, c, \dots\}$ be a set of constants. P represents a channel name from the set $\mathcal{L} = \mathbb{N} \cup \mathcal{S} \cup (\mathcal{E}, \mathcal{S})$, where $m, n, \dots$ denote numerical labels taken from $\mathbb{N}$. $(\mathcal{E}, \mathcal{S})$ is a pair where the second member must be in $\mathcal{S} = \{p, q, \dots\}$ which is a set of symbolic labels and the first member must be in $\mathcal{E} = \{\alpha, \beta, \gamma, \dots\}$ which is a special set of labels called *environment labels.* $\mathbb{N} \cap S = \mathcal{E} \cap S = \mathbb{N} \cap \mathcal{E} = \emptyset$. 0 represents the null process. In our extension we introduce the notion of *environment.* A channel can be represented by numeric labels or by pairs of an environment label and a symbolic label. The environment label is a special label which can be used as a term. Also the environment label can be empty. Symbol ω represents the channel.

The language of the formal system is $\mathcal{M}$. It is defined as follows:

$$\begin{aligned} M &::= 0 \mid a \mid \alpha \mid x \mid \lambda_P x.M \mid \widehat{\ }_P M \mid M \parallel M \mid M \circ M \mid (M + M) \mid M! \mid \lambda_\omega x.M \\ P &::= \text{number} \mid \text{symbol} \end{aligned}$$

The terms of this signature are called λ_{MEP}-terms. In concurrent programming, the terms of $\mathcal{M}$ represent processes. Operation symbols $\circ, \parallel, +, !$ are considered as process constructors. The communication $\lambda_P x.$ (abstraction or input) and $\widehat{\ }_P$ (parameter passing or output) are also considered as process constructors, whose effect over a process M is to define processes ($\lambda_P x.M$ and $\widehat{\ }_P M$, respectively) which are involved in functional applications under certain given conditions. Constants, variables and the null process form the atomic symbols of the language. The interpretation of the process constructors is as follows. *Sequential composition* ($\circ$): it defines a process $M \circ N$ from the processes M and N. The process N is executed after M. *Parallel composition* ($\parallel$): it defines a new process $M \parallel N$ from the processes M and N. Both processes are executed simultaneously. *Non-deterministic choice* ($+$): $M + N$ shows the capability of M or N (either one or the other) to take part in a communication. A communication can be made by one of them and the other disappears. *Replicator* (!): $!M$ defines a process which produces the multiple generation of M. It acts as a warehouse, where it is possible to take an infinite number of copies of the process M. Operators have the subsequent priority order: $! \ > \ \widehat{\ }_P \ > \ \circ \ > \ \lambda_P x. \ > \ + \ > \ \parallel$.

An occurrence of the variable x is *bound* in a process M iff it is in a term $\lambda_P x.M$ (for all $P \in \mathcal{L}$). In any other case, the occurrence of x is *free.* If x has at least one free occurrence in M, we say that x is a *free* variable of M. We denote $FV(M)$ the set of free variables of M. A process is *closed* if it does not have free variables. A program is a closed process.

The relations between processes are described in terms of communications. Constructors $\lambda_P x.$ and $\widehat{\ }_P$ provide processes M and N with the possibility of communicating through a channel labeled with $P \in \mathcal{L}$ as follows. *Constructor of input* ($\lambda_P x.$): through channel P process $\lambda_P x.M$ can receive a process which replaces the free occurrences of x in M by the incoming process through this channel. *Constructor of output* ($\widehat{\ }_P$): process $\widehat{\ }_P M$ can send a process M through channel P. It then becomes inactive. In addition, a partial order relation is defined on the set of labels which we denote as $\preceq_{\mathcal{L}}$. It is required as a condition for the relation that the numerical label 0 be the minimum of the ordered set. We use the partial order relation in the set of labels to represent the application order of the real parameters to the formal parameters of a function, which is analogous to how it is done in the λ-calculus in order to treat the problem of parameterization in function calls. To represent the ordering in channels of the third type, (pairs environment-label) we consider to apply the partial order over channels with the same environment applying the partial ordering over the symbolic label.

Reduction rules. We can define the β-comunication rules which describe the evolution of a pair of *communicating* terms when the processes are composed in sequential or parallel mode [9]. β_{sec} (Sequential β-comunication) $(\cdots + \lambda_P x.M) \circ (\widehat{{}_P} N \circ L + \cdots) \longrightarrow_{\beta_{sec}} M[N/x] \circ L$, $P \in \mathcal{L} - \{0\}$ and β_P (Parallel β-comunication, through P) $(\cdots + \lambda_P x.M) \parallel (\widehat{{}_P} N \circ L + \cdots) \longrightarrow_{\beta_P} M[N/x] \parallel L$, $P \in \mathcal{L} - \{0\}$ where P is a channel name from the set $\mathcal{L} = \mathbb{N} \cup (\mathcal{E}, \mathcal{S}) \cup \mathcal{S}$.

Inference rules. The reduction rules and the re-ordering rules ([9]) are the axioms of the formal theory. Let us define the reduction rule → over the processes of the language $\mathcal{M}$. The inference rules of this formal system which show the computational semantics of the LCMEP operators are [8]:

$$\mu_{\circ_1} : \frac{M \rightarrow M'}{M \circ N \rightarrow M' \circ N} \qquad \mu_{\circ_2} : \frac{M \rightarrow M'}{N \circ M \rightarrow N \circ M'}$$

$$\mu_{\lambda} : \frac{M \rightarrow M'}{\lambda_p x.M \rightarrow \lambda_p x.M'} \qquad \mu_{\widehat{\ }} : \frac{M \rightarrow M'}{\widehat{{}_p} M \rightarrow \widehat{{}_p} M'}$$

$$\mu_{\parallel} : \frac{M \rightarrow M'}{M \parallel N \rightarrow M' \parallel N} \qquad \mu_{Struct} : \frac{M \equiv N \quad N \rightarrow N' \quad N' \equiv M'}{M \rightarrow M'}$$

For a more complete explanation of LCMEP, see [7].

3 The Translation

In this section, we present a phrase-by-phrase translation of a (simple version of a) parallel object-oriented programming language (à la POOL [2]) into LCMEP (see [7] for details). There, we show a complete example of the use of the language. Each syntactical entity is represented as a λ_{MEP}-term and composite entities are represented by their components. LCMEP cannot use multiple parameters as channels. However, we can communicate through numeric channels (using currying we can model polyadic functions) or by sending some processes in parallel. This problem has an easy treatment in the *polyadic* π-calculus [8] by sending multiple names through the channels. In LCMEP, P represents a channel through which a process N is sent from an output $\widehat{{}_P} N$ into an input $\lambda_p x.M$, where the occurrences of x in M are replaced by the process N, i.e. every valid λ_{MEP}-term (variables, constants, processes, environments, etc. but not channels). The idea of conversion is to use the environment name to substitute the agent name. When we need to send a name, we actually send an *environment* process through the channel.

In this paper, the process perspective is presented. Then, the definition of classes comes from the definition of the objects' behaviour. To simplify the notation, construction blocks are used: *Value*(l, v) represents the evaluation of an expression or variable $(\widehat{{}_l}\, v)$. *Null*(l) is translated as an *output* through channel l

of the *null* term $(\widehat{_{l}}\,0)$. $Wait(l, l')$ is an abstraction (waiting) and a later evaluation $(\lambda_{l'} v.\, Value(l, v))$. $Result(l, l', l'')$ is its completion $(\lambda_{l''} v.\, Value(l', v) \circ Null(l))$.

Simple POOL expressions are translated into LCMEP as follows:

$$\begin{array}{ll} [new(B)](l, class, local) & \equiv \widehat{_{class}}(local) \circ \lambda_{(local,new)} x.\, Value(l, x) \\ [self](l, v) & \equiv Value(l, v) \end{array}$$

$$\begin{array}{ll} [Y](l, y) & \equiv \widehat{_{y}}(\widehat{_{r}}\ 0) \circ \lambda_y v.\, Value(l, v) + \widehat{_{y}}(\widehat{_{u}}\ v') \circ \lambda_y v.\, Value(l, v) \\ [Var\ Y](y, v) & \equiv Var(y, v) \\ Var(y, v) & \equiv \lambda_y x_1.(x_1 \parallel \lambda_r x.(\widehat{_{y}}\ v \circ Var(y, v)) + \\ & \quad +\lambda_u v'.(\widehat{_{y}}\ v' \circ Var(y, v'))) \end{array}$$

The creation of a new object is modeled as a *call* to the class sending the local environment label and the later reception of the new object identifier (its *public environment*) through this local environment and the "new label" as a channel. That object identifier must be a label from the *environment set*. Using a variable is equivalent to updating it or to consulting it. In any case, the declaration of a variable is the choice of one of both answers (For simplicity we do not use in this case the environment-label channel but in the actual translation we always use that form).

POOL classes' declaration in LCMEP is:

$$\begin{array}{ll} [Cdec](class) & \equiv !\ \lambda_{class} ret.(\lambda_\omega x.\lambda_\omega y.(\widehat{_{(ret,new)}}\ x \parallel Newobject(x, y))) \\ Newobject(\alpha_i, \epsilon_i) & \equiv ([S] \parallel [Vdecs](\epsilon_i, \widetilde{y}, \widetilde{v}) \parallel [Mdecs](\epsilon_i, \widetilde{m})) \\ [Mdecs](\epsilon_i, \widetilde{m}) & \equiv [Mdec_1](\epsilon_i, m_1) \parallel \cdots \parallel [Mdec_i](\epsilon_i, m_n) \end{array}$$

Class declaration is a replicator of a term which contains a reception of a message with the environment of the sender. The term takes two new environments "x" and "y" from $\mathcal{E}$ and returns first the original sender through the channel "new" and considering the second one as the private environment. After this, it creates a new object with these environments. For simplicity, local communications through channels are represented labeled by l_i. Actually we always must use channels as pairs (ϵ, l_i) where ϵ is the private environment of each object. In addition, we are going to make the synchronous request for the execution of a remote method into LCMEP. First, let us see the translation of an object α_0 which requests the execution of a remote method m with only one evaluated parameter.

$$[\beta_0!M(\beta_1)](l, \alpha_0, m) \equiv \lambda_\omega safe.(\widehat{_{(\alpha_0,m)}} safe \parallel (\widehat{_{(safe,m)}}\ \beta_1 \circ Wait(l, (safe, m))))$$

In this case, a term which contains a secure channel is sent to the object (received by an *answer* into the object body); after this, it sends the parameters and then the object waits using $Wait(l, l')$.

The communication in POOL is synchronous but it is possible to use asynchronous communication by using the following result. The declaration of a one-parameter method is:

$$\begin{aligned}
[\mathit{Mdec}](\epsilon,m) &\equiv\, !\ \mathit{Mmethod}(\epsilon,m)\\
\mathit{Mmethod}(\epsilon,m) &\equiv \mathit{Mhandle}(\epsilon,m) \parallel [\mathit{Var}\ Y_1](\epsilon,y_1,v_1) \parallel \lambda_{(\epsilon,g)}x.[E](\epsilon,m)\\
\mathit{Mhandle}(\epsilon,m) &\equiv \lambda_{(\epsilon,m)}\mathit{safe}.(\lambda_{(safe,m)}x._{\widehat{(\epsilon,y_1)}}(_{\widehat{(\epsilon,u)}}\ x) \circ \lambda_{(\epsilon,y_1)}x.0 \circ (_{\widehat{(\epsilon,g)}}\ 0 \parallel\\
&\quad \parallel \mathit{Result}((\epsilon,\mathit{ans}),(\mathit{safe},m),(\epsilon,m))))
\end{aligned}$$

If we want to establish an asynchronous communication, we cannot only move the position of the *Result*. If we do that, we could active the *answer* term before we want. Therefore, the *Result* term is separated in two components: first, a term which awakes the *Wait* in the client object; second, a term which is placed instead of the Result and awakes the *answer.*

$$\begin{aligned}
[\mathit{Mdec}](\epsilon,m) &\equiv\, !\ \mathit{Mmethod}(\epsilon,m)\\
\mathit{Mmethod}(\epsilon,m) &\equiv \mathit{Mhandle}(\epsilon,m) \parallel [\mathit{Var}\ Y_1](\epsilon,y_1,v_1) \parallel \lambda\ _{(\epsilon,g)}x.[E](\epsilon,m)\\
\mathit{Mhandle}(\epsilon,m) &\equiv \lambda_{(\epsilon,m)}\mathit{safe}.(\lambda_{(safe,m)}x._{\widehat{(\epsilon,y_1)}}(_{\widehat{(\epsilon,u)}}\ x) \circ \lambda_{(\epsilon,y_1)}x.0 \circ\\
&\quad \circ\ \mathit{Value}((\mathit{safe},m),0) \circ (_{\widehat{(\epsilon,g)}}\ 0 \parallel \lambda_{(\epsilon,m)}\mathit{aux}.\mathit{Null}(\epsilon,\mathit{ans})))
\end{aligned}$$

There are no more differences between synchronous and asynchronous methods. Actually, the operational effect is to alter the order of terms. To save space and to preserve simplicity, we only present the synchronous communication. However, examples of synchronous and asynchronous communication are in [7]. The receipt of a parameter is translated by updating the variable Y_1, joined to the parallel composition with the answer sender term and another that synchronizes with the evaluation of the method body. Now let us see a similar process, but using multiple parameters:

$$\begin{aligned}
[\beta_0!M(\beta_1,\ldots,\beta_n)](l,\alpha_0,m) \equiv\ &\lambda_\omega \mathit{safe}.(_{\widehat{(\alpha_0,m)}}\mathit{safe} \parallel\ _{\widehat{(safe,m)}}\ \beta_1 \circ\cdots\circ\\
&\circ\ _{\widehat{(safe,m)}}\ \beta_n \circ \mathit{Wait}(l,(\mathit{safe},m))
\end{aligned}$$

The method declaration into the server object is:

$$\begin{aligned}
[\mathit{Mdec}](\epsilon,m) &\equiv\, !\ \mathit{Mmethod}(\epsilon,m)\\
\mathit{Mmethod}(\epsilon,m) &\equiv \mathit{Mhandle}(\epsilon,m) \parallel [\mathit{Var}\ Y_1](y_1,v_1) \parallel \ldots [\mathit{Var}\ Y_n](y_n,v_n) \parallel\\
&\quad \parallel \lambda_{(\epsilon,g)}x.[E](\epsilon,m)\\
\mathit{Mhandle}(\epsilon,m) &\equiv \lambda_{(\epsilon,m)}\mathit{safe}.(\lambda_{(safe,m)}x._{\widehat{(\epsilon,y_1)}}(_{\widehat{(\epsilon,u)}}\ x) \circ \lambda_{(\epsilon,y_1)}x.0 \circ\\
&\quad \circ\cdots\circ \lambda_{(safe,m)}x._{\widehat{(\epsilon,y_n)}}(_{\widehat{(\epsilon,u)}}\ x) \circ \lambda_{(\epsilon,y_n)}x.0\circ\\
&\quad \circ (_{\widehat{(\epsilon,g)}}\ 0 \parallel \mathit{Result}((\epsilon,\mathit{ans}),(\mathit{safe},m),(\epsilon,m)))
\end{aligned}$$

If instead of the evaluated parameters there are expressions (knowing that a POOL expression must be left-to-right evaluated), it is required that the evaluation of an expression E_i do not start until the evaluation of the expresion E_{i-1} has finished. This effect is successfully represented by:

$$\begin{aligned}
&[\beta_0!M(\beta_1,\ldots,\beta_{i-1},E_i,\ldots,E_n)](l,\alpha_0,m)\ \equiv\\
&\quad \lambda_\omega \mathit{safe}.(_{\widehat{(\alpha_0,m)}}\mathit{safe} \parallel\ _{\widehat{(safe,m)}}\ \beta_1 \circ\cdots\circ\ _{\widehat{(safe,m)}}\ \beta_{i-1} \circ \lambda_{l_i}\beta_i._{\widehat{(safe,m)}}\ \beta_i\\
&\quad \circ\ _{\widehat{(\epsilon,g_{i+1})}}\ 0 \circ\cdots\circ \lambda_{l_n}\beta_n._{\widehat{(safe,m)}}\ \beta_n \circ \mathit{Wait}(l,(\mathit{safe},m)) \parallel [E_i](l_1) \parallel\\
&\quad \lambda_{(\epsilon,g_{i+1})}x.[E_{i+1}](l_{i+1}) \parallel \cdots \parallel \lambda_{(\epsilon,g_n)}x.[E_n](l_n)
\end{aligned}$$

In our proposal, we do not include the channels of the parameters because in most cases they are basically local.

The treatment of a *local call* may be similar to the treatment of a *remote call.* Actually, it only differs in one feature. In a local call, a λ_{MEP}-term is sent which contains the safe environment to communication. But it is not received by the answer. So no one receives the communication of the result to the answer. Therefore, a new term is added which gets the answer role. A multiple evaluated arguments method is as follows:

$$\begin{aligned}[M(\beta_1,\ldots,\beta_n)](\epsilon,m,l) \equiv \lambda_\omega \mathit{safe}.(\widehat{_{(\epsilon,m)}}\,\mathit{safe} \circ \widehat{_{(\mathit{safe},m)}}\ \beta_1 \circ \cdots \circ \widehat{_{(\mathit{safe},m)}}\ \beta_n \circ \\ \circ\, (\lambda_{(\epsilon,ans)} y.0 \parallel \mathit{Wait}(l,(\mathit{safe},m))))\end{aligned}$$

If there are evaluated parameters and expressions then:

$$\begin{aligned}&[M(\beta_1,\ldots,\beta_{i-1},E_i,\ldots,E_n)](m,l)\ \equiv \\ &\quad \lambda_\omega \mathit{safe}.(\widehat{_{(\epsilon,m)}}\ \alpha_0 \circ \widehat{_{(\mathit{safe},m)}}\beta_1 \circ \cdots \circ \widehat{_{(\mathit{safe},m)}}\ \beta_{i-1} \circ \lambda_{l_i}\beta_i\cdot\widehat{_{(\mathit{safe},m)}}\ \beta_i \circ \\ &\quad \widehat{_{(\epsilon,g_{i+1})}}\ 0 \circ \cdots \circ \lambda_{l_n}\beta_n\cdot\widehat{_{(\mathit{safe},m)}}\ \beta_n \circ (\lambda_{(\epsilon,ans)} y.0\, \mathit{Wait}(l,(\mathit{safe},m))) \parallel \\ &\quad \parallel [E_i](l_i) \parallel \lambda_{(\epsilon,g_{i+1})} x.[E_{i+1}](l_{i+1}) \parallel \cdots \parallel \lambda_{(\epsilon,g_n)} x.[E_n](l_n)))\end{aligned}$$

For a local declaration of variables into an expression the translation is the parallel composition of the expression and the variables. If there are local variables into an object then channels are pairs environment-label. For increasing the simplicity of the translation we usually call y the actual pair (ϵ,y).

Now, let us see the translation into LCMEP of the auxiliary expressions whose definitions were postponed: $[\mathit{Wait}(\beta)](l,l') \equiv \mathit{Wait}(l,l')$, $[E \Rightarrow \beta](l,\ldots) \equiv [E](l',\ldots) \parallel \mathit{Result}(l,l',l'')$, $[E,\rho](l,\ldots) \equiv [E](l,\ldots) \parallel [\rho](\widetilde{y},\widetilde{v})$, $[\rho](\widetilde{y},\widetilde{v}) \equiv \mathit{Var}(y_1,v_1) \parallel \cdots \parallel \mathit{Var}(y_n,v_n)$, $[nil](l) \equiv \mathit{Null}(l)$.

The assignment is expressed as the parallel composition of two λ_{MEP}-terms: one which evaluates the expression and sends the result through the auxiliary channel l', and the other which receives the result of the expression from this channel and updates the variable Y by assigning the value of the expression.

With regard to the *answer* which replies to the request of remote invocation of methods, $[E_0!M(E_1,\ldots,E_n)]$, we can see that it accepts any request of any available method with a secure channel from the environment set. Sending it allows for the replicator of the method to execute a copy of itself. It returns the result (to activate the client process which is in a *wait* statement) and later returns a *null* through its own channel l to finish.

$$\begin{aligned}&[Y := E](l,\ldots,y) \equiv [E](l',\ldots) \parallel \lambda_{l'} v'.(\widehat{_y}(\widehat{_u}\ v') \circ \lambda_y v.0) \\ &[\mathit{answer}(m_1,\ldots,m_p)](\alpha,\epsilon,m_1,\ldots,m_p,l) \equiv \\ &\quad (\lambda_{(\alpha,m_1)} x.(\widehat{_{(\epsilon,m_1)}} x + \cdots + \lambda_{(\alpha,m_p)} x.(\widehat{_{(\epsilon,m_p)}} x) \circ \lambda_{(\epsilon,ans)} y.\mathit{Null}(l))\end{aligned}$$

Now, we introduce the description of the basic expression of the standard class *Bool* since this standard class is needed for the definition of the *conditional* and *loop* statements (the full standard class *Boolean* is available in [7]):

$$\begin{aligned}[\mathit{true}](l,\epsilon,\mathit{bool}) &\equiv \widehat{_{(\mathit{bool},\mathit{val})}}\epsilon \circ \widehat{_{(\epsilon,\mathit{true})}}0 \circ \mathit{Null}(l) \\ [\mathit{false}](l,\epsilon,\mathit{bool}) &\equiv \widehat{_{(\mathit{bool},\mathit{val})}}\epsilon \circ \widehat{_{(\epsilon,\mathit{false})}}0 \circ \mathit{Null}(l)\end{aligned}$$

Finally, modeling the conditional and iterative statements is as follows:

$$\begin{aligned}
[\mathit{if}\ \ E\ \ \mathit{then}\ S_1\ \ \mathit{else}\ S_2](l) &\equiv \mathit{BoolEval}(l', \epsilon, l_1, l_2) \parallel [E](l') \parallel \lambda_{l_1} x.[S_1](l)\\
&\quad +\lambda_{l_2} x.[S_2](l)\\
\mathit{BoolEval}(l', \epsilon, l_1, l_2) &\equiv \lambda_{l'} x.(x \parallel \lambda_{(\epsilon,true)} x.\widehat{l_1}\ 0 + \lambda_{(\epsilon,false)} x.\widehat{l_2}\ 0)\\
[\mathit{while}\ \ E\ \ \mathit{do}\ \ S](l, \dots) &\equiv \mathit{Loop}(l, l', \dots) \parallel\ !\ \lambda_{l'} v.\mathit{Loop}(l, l', \dots)\\
\mathit{Loop}(l, l', \dots) &\equiv (\mathit{BoolEval}(l'', \epsilon, l_1, l_2) \parallel [E](l'') \parallel\\
&\quad \parallel \lambda_{l_1} x.[S](l', \dots) + \lambda_{l_2} x.\mathit{Value}(l, v))
\end{aligned}$$

4 Conclusions and Future Work

An extension of LCEP has been briefly presented, LCMEP. Under a process perspective, this extension can model a simple version of a parallel object-oriented language, POOL. The phrase-by-phrase translation process has been made by modeling object definition, creation, management, communication, and evolution features. The object activities are synchronized by sending messages which contain references to other system objects. Among the characteristics of LCEP, we can not find the sending of communicating channels. In our extension into LCMEP a special class of labels with two functions is considered: the environment function, and the term function. As an environment we consider the label as the first member of a pair $(\mathcal{E}, \mathcal{S})$ which restricts the communication between terms whith the same environment. As channel, an environment label can be sent to another term to communicate the environment of the sender. These features eliminate the obstacles for the modeling of the behaviour of the chosen POOL variant. As we have shown, LCMEP is an effective tool which has sucessfully been used to model parallel processes as well as to model object-oriented language features. A translator system from a high level language (ALEPH) into LCEP is already available [5]. By transferring the proposal extension LCMEP to that system, we will obtain an explicitly parallel high-level language with object-oriented features.

References

1. H. Aït-Kaci and J. Garrigue. Label-Selective λ-Calculus: Syntax and Confluence. In *Proc. of the 13th Int. Conf. on Foundations of Software Technology and T.C.S.*, volume 761 of *LNCS*. Springer-Verlag, Berlin, 1993. 396
2. P. America, J. de Bakker, J. Kok, and J. Rutten. Operational Semantics of a Parallel Object-Oriented language. *In Proc. of the 13th Symposium on Principles of Programming Languages*, pages 194–208, 1986. 395, 398
3. P. America and J. Rutten. *A parallel object-oriented language: design and semantic foundations*, pages 1–49. Willey Series in Parallel Computing. J. W. de Bakker, 1990. 395, 395
4. G. Berry and G. Boudol. The Chemical Abstract Machine. *In Proc. of 20th ACM Annual Symp. on P. P. L.*, pages 81–93, ACM Press, 1993. 396
5. L. Climent, M. L. Llorens, and J. Oliver. Building an interpreter for label-selective λ-calculus. In B. Clares, editor, *Proc. of II Jornadas de Informática*, pages 325–334. Asociación Española de Informática y Automática, Almuñecar (Spain), 1996. 402

6. C. Herrero and J. Oliver. Object-oriented parallel label-selective λ-calculus. *BRICS Notes Series (NS-98-5)*, pages 7–15, 1998. 396
7. C. Herrero and J. Oliver. Parallel multi-label-selective λ-Calculus: An object-oriented proposal. Technical Report DSIC-II/10/99, UPV, 1999. 396, 398, 398, 400, 401
8. R. Milner. The polyadic π-calculus: A tutorial. In F. L. Brauer, W. Bauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specifications*. Springer-Verlag, Berlin, 1993. 398, 398
9. J. Oliver. *Extension of λ-calculus for Modelling Concurrent Processes*. PhD thesis, DSIC (UPV), 1996. 396, 396, 398, 398
10. D. Walker. Objects in the π-calculus. In *Information and Computation*, volume 116, pages 253–271. 1995. 396

Simulation Problems for One-Counter Machines*

Petr Jančar[1], Faron Moller[2], and Zdeněk Sawa[1]

[1] Technical University of Ostrava, Czech Republic
[2] Uppsala University, Sweden

Abstract. We consider decidability questions for simulation preorder (and equivalence) for processes generated by one-counter machines. We sketch a proof of decidability in the case when testing for zero is not possible, and demonstrate the undecidability in the general case.

1 Introduction

A ***one-counter machine*** is a nondeterministic finite-state automaton acting on a single counter variable ranging over the set $\mathbb{N}$ of nonnegative integers. Formally, it is a tuple $M = \langle Q, \Sigma, \delta^{=}, \delta^{>} \rangle$ where Q is a finite set of ***control states***, Σ is a finite set of ***actions***, and $\delta^{=}, \delta^{>} : Q \times \Sigma \to \mathcal{P}(Q \times \{-1, 0, 1\})$ are ***transition functions*** (where $\mathcal{P}(A)$ denotes the set of subsets of A). To M we associate the transition system $\langle \Gamma, \{\xrightarrow{a}\}_{a \in \Sigma} \rangle$, where $\Gamma = \{\, p(n) \mid p \in Q, n \in \mathbb{N} \,\}$ is the set of ***states*** and each $\xrightarrow{a} \subseteq \Gamma \times \Gamma$ is a binary relation defined as follows:

$$p(n) \xrightarrow{a} p'(n+i) \quad \text{iff} \quad \begin{cases} n = 0,\ i \geq 0,\ \text{and } (p', i) \in \delta^{=}(p, a); \quad \textit{or} \\ n > 0,\ \text{and } (p', i) \in \delta^{>}(p, a) \end{cases}$$

Note that any transition increments, decrements, or leaves unchanged the counter value; and a decrementing transition is only possible if the counter value is positive. Also observe that when $n > 0$ the transitions of $p(n)$ do not depend on the actual value of n.

M is ***deterministic*** iff for any state $p(n)$ and for any action $a \in \Sigma$ there is at most one state $p'(n')$ such that $p(n) \xrightarrow{a} p'(n')$. M is a ***weak one-counter machine*** iff $\delta^{=} = \delta^{>}$. Thus, a weak one-counter machine can test if its counter is nonzero (that is, it can perform certain transitions on the proviso that its counter is nonzero), but it cannot test if its counter is zero.

A binary relation $\mathcal{S}$ between the states of two (weak) one-counter machines is a ***simulation*** iff, given $\langle p(m), q(n) \rangle \in \mathcal{S}$ and $p(m) \xrightarrow{a} p'(m')$, we have $q(n) \xrightarrow{a} q'(n')$ with $\langle p'(m'), q'(n') \rangle \in \mathcal{S}$. $p(m)$ is ***simulated*** by $q(n)$, written $p(m) \preccurlyeq q(n)$, iff they are related by some simulation relation $\mathcal{S}$; $p(m)$ and $q(n)$ are ***simulation equivalent***, written $p(m) \simeq q(n)$, iff $p(m) \preccurlyeq q(n)$ and $q(n) \preccurlyeq p(m)$. If two states are related by a symmetric simulation relation, then they are ***bisimilar***.

* The first and third authors are partially supported by the Grant Agency of the Czech Republic, Grant No. 201/97/0456. The second author is partially supported by TFR grant No. 221-98-103.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 404–413, 1999.

Such automata, and behavioural concepts like (bi)simulation equivalence, are enjoying renewed interest within the automata and process theory communities due to the present active search for the dividing line between decidable and undecidable theories for classes of infinite state systems (see, e.g., [4]). Recently, Abdulla and Čerāns [1] outlined an extensive and involved proof of the decidability of simulation preorder over weak one-counter machines. Their 16-page extended abstract is very technical and omits the proofs of most of the crucial lemmas. Here we outline a short proof based on simple and intuitive ideas about colouring the plane. (Due to space limitations and further results communicated here, we omit the proofs of some technical lemmas; for these, we refer to the following ten page report [3].)

We then show that simulation preorder between deterministic one-counter machines is undecidable; to do this we use a reduction from the halting problem for Minsky machines. Finally we demonstrate the undecidability of simulation equivalence in the general case; this contrasts with the decidability of bisimulation equivalence [2].

2 Decidability for Weak One-Counter Machines

For any pair of control states $\langle p, q\rangle \in Q_1 \times Q_2$ taken from two weak one-counter machines, we can ask for what values $m, n \in \mathbb{N}$ do we have $p(m) \preccurlyeq q(n)$. We can picture the "graphs" of the functions $\mathbb{G}_{\langle p,q\rangle} : \mathbb{N} \times \mathbb{N} \to \{\text{black}, \text{white}\}$ given by

$$\mathbb{G}_{\langle p,q\rangle}(m,n) = \begin{cases} \text{black, if} & p(m) \preccurlyeq q(n); \\ \text{white, if} & p(m) \not\preccurlyeq q(n) \end{cases}$$

by appropriately colouring (black or white) the integral points in the first quadrant of the plane. Note that if $p(m) \preccurlyeq q(n)$ then $p(m') \preccurlyeq q(n')$ for all $m' \leq m$ and $n' \geq n$; hence the black points are upwards- and leftwards-closed, and the white points are downwards- and rightwards-closed. For a fixed pair of states $p_0(m_0)$ and $q_0(n_0)$ of these weak one-counter machines, we decide the question "Is $p_0(m_0) \preccurlyeq q_0(n_0)$?" by effectively constructing an initial portion of the $|Q_1| \times |Q_2|$ graphs which includes the point $\langle m_0, n_0\rangle$, and look to the colour of $\mathbb{G}_{\langle p_0,q_0\rangle}(m_0,n_0)$.

Define the ***frontier function*** $f_{\langle p,q\rangle}(n) = \max\{m : \mathbb{G}_{\langle p,q\rangle}(m,n) = \text{black}\}$, that is, the greatest value m such that $p(m) \preccurlyeq q(n)$; $f_{\langle p,q\rangle}(n) = \infty$ if $\mathbb{G}_{\langle p,q\rangle}(m,n) = \text{black}$ for all m; and $f_{\langle p,q\rangle}(n) = -1$ if $\mathbb{G}_{\langle p,q\rangle}(0,n) = \text{white}$. This function is monotone nondecreasing, and the set of ***frontier points*** $\langle f_{\langle p,q\rangle}(n), n\rangle \in \mathbb{N} \times \mathbb{N}$ defines the ***frontier*** of $\mathbb{G}_{\langle p,q\rangle}$, the collection of the rightmost black points from each level. Slightly abusing notation, we use f to refer to the frontier function as well as the frontier given by the frontier function. The next theorem is the clue to our decidability result.

Belt Theorem
Every frontier lies in a linear belt with rational (or ∞) slope.

The proof of the Belt Theorem is outlined in Section 4; in the remainder of this section we describe the decision procedure which is based on this theorem. For each $k = 0, 1, 2, \ldots$, let $\mathbb{G}^k_{\langle p,q\rangle} : \mathbb{N} \times \mathbb{N} \to \{\text{black, white}\}$ be the maximally-black collection of colourings of the plane (with $\langle p, q\rangle$ ranging over $Q_1 \times Q_2$) which satisfies the following: whenever $m, n \leq k$ with $\mathbb{G}^k_{\langle p,q\rangle}(m, n) = \text{black}$, if $p(m) \xrightarrow{a} p'(m')$ then $q(n) \xrightarrow{a} q'(n')$ with $\mathbb{G}^k_{\langle p',q'\rangle}(m', n') = \text{black}$. Such a maximally-black collection of colourings exists since the collection of totally-white colourings satisfies this condition, as does the collection of colourings which colours a point black exactly when it is black in *some* collection of colourings satisfying the above condition; this final collection of colourings is the one we seek, and is effectively computable (it is black everywhere outside the initial $k \times k$ square). Note that $\mathbb{G}^k_{\langle p,q\rangle}(m, n) = \text{white}$ implies $\mathbb{G}^{k+1}_{\langle p,q\rangle}(m, n) = \text{white}$ and that $\mathbb{G}_{\langle p,q\rangle} = \lim_{k\to\infty} \mathbb{G}^k_{\langle p,q\rangle}$.

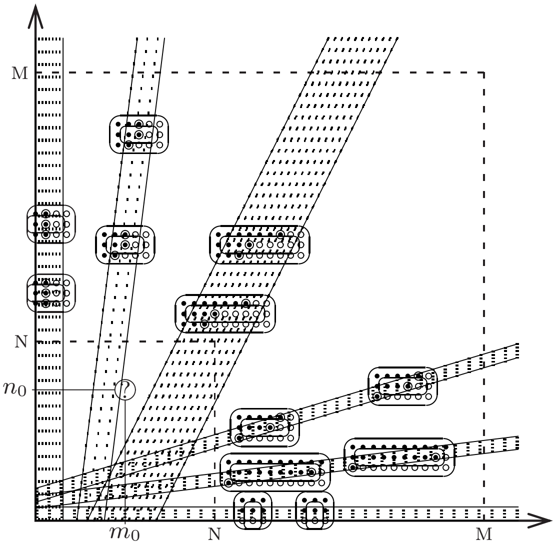


Fig. 1.

We can effectively construct the collections $\mathbb{G}^k_{\langle p,q\rangle}$ for each $k = 1, 2, 3, \ldots$. By the Belt Theorem, eventually for some k we must be able to lay down over the graphs $\mathbb{G}^k_{\langle p,q\rangle}$ a set of linear belts with rational slopes such that (see Fig. 1)

- there is an initial (M × M) square (M $< k$) inside of which each frontier lies within some belt (we may assume that parallel belts coincide, so that two or more frontiers may appear in the same belt);
- outside of some initial (N × N) square (N $<$ M) containing the point $\langle m_0, n_0\rangle$, the belts are separated by gaps wide enough so that no point has neighbouring points in two belts;

- within the area bounded by the initial (N × N) and (M × M) squares, looking at each horizontal level within each belt (or each vertical level, in the case of a horizontal belt) we find a pattern which repeat itself—along with all of its neighbouring points—at two different levels. (That is, the colourings of the points and neighbouring points are the same in every graph on these levels within the belt.) Furthermore, the shift from one occurrence of the pattern to the next has a slope equal to that of the belt.

Note that these belts need not *a priori* be the true frontier belts specified in the Belt Theorem; but since (by the pigeonhole principle) the true frontier belts display such a repetitive pattern, the true frontier belts must eventually appear in the above fashion if no other belts appear earlier on in the construction.

Once we recognise such belts in the graphs $\mathbb{G}^k_{\langle p,q\rangle}$ (for some k), we can define graphs $\mathbb{G}'_{\langle p,q\rangle}$ by continuing the colouring of the graphs $\mathbb{G}^k_{\langle p,q\rangle}$ by periodically repeating the colouring within the belts between the levels at which the patterns repeat, and recolouring points to the right of the belts to maintain the invariant that white points are rightwards-closed.

We can readily confirm that the set of all pairs $\langle p(m), q(n)\rangle$ such that $\mathbb{G}'_{\langle p,q\rangle}(m,n)$ is black is a simulation. Thus, all black points are correct (that is, $\mathbb{G}_{\langle p,q\rangle}(m,n)$ is black whenever $\mathbb{G}'_{\langle p,q\rangle}(m,n)$ is black), and all white points within the initial (N × N) square are correct, proving that we have correctly constructed the initial (N × N) square.

3 Undecidability for One-Counter Machines

To show undecidability, we use a reduction from the halting problem for Minsky machines with 2 counters, which is well-known to be undecidable [5]; we can even suppose the input counter values to be zero. We use the following definition:

A ***Minsky machine*** C with two nonnegative counters c_1, c_2 is a program

$$1\colon \mathit{COMM}_1;\;\; 2\colon \mathit{COMM}_2;\;\; \cdots\;\; ;\;\; n\colon \mathit{COMM}_n$$

where COMM_n is a *halt*-command and COMM_i ($i = 1, 2, \ldots, n-1$) are commands of the following two types (assuming $1 \le k, k_1, k_2 \le n, 1 \le j \le 2$)

(1) $c_j := c_j + 1;\; \mathit{goto}\ k\ [\mathit{action}\ \mathbf{i}_j]$
(2) $\mathit{if}\ c_j = 0\ \mathit{then\ goto}\ k_1\ [\mathit{action}\ \mathbf{z}_j]\ \mathit{else}\ (c_j := c_j - 1;\ \mathit{goto}\ k_2\ [\mathit{action}\ \mathbf{d}_j])$

Note that the computation of the machine C (starting with COMM_0, the counters initialized to 0) corresponds to a sequence of actions from $\{\mathbf{i}_1, \mathbf{i}_2, \mathbf{z}_1, \mathbf{z}_2, \mathbf{d}_1, \mathbf{d}_2\}$. This sequence is finite if the computation of C halts and infinite otherwise.

Theorem 1. *The problem if $p(0) \preccurlyeq q(0)$, where $p(0)$ and $q(0)$ are states of deterministic one-counter machines M_1 and M_2, is undecidable. (This holds even in restricted cases, where one of M_1 and M_2 is fixed.)*

Proof. Given a Minsky machine C with 2 counters c_1, c_2, we describe the construction of two deterministic one-counter machines M_1 and M_2, with specified control states p, q respectively, such that $p(0) \preccurlyeq q(0)$ iff C does not halt. In an obvious way, C can be transformed to a deterministic one-counter machine M_1 with n control states (n being the number of commands of C), with the set of actions $\{\mathbf{i}_1, \mathbf{i}_2, \mathbf{z}_1, \mathbf{z}_2, \mathbf{d}_1, \mathbf{d}_2\}$, and such that its counter 'behaves' like c_1 (actions $\mathbf{i}_1, \mathbf{z}_1, \mathbf{d}_1$ depend on and change it) while the actions $\mathbf{i}_2, \mathbf{z}_2, \mathbf{d}_2$ ignore the counter (c_2 is 'missing'). Thus a computation of M_1 can digress from that of C by performing an action $\mathbf{d}_2$ instead of $\mathbf{z}_2$ or vice versa. M_2 can be constructed similarly, now with the counter corresponding to c_2 while c_1 is ignored. Moreover, for each control state of M_2 with 'outgoing arcs' labelled $\mathbf{z}_2$ (enabled when the counter is zero) and $\mathbf{d}_2$ (enabled when the counter is positive) we add new 'complementary' arcs labelled $\mathbf{z}_2$ (for positive) and $\mathbf{d}_2$ (for zero) which lead to a special control state q_* which has a loop for any action (ignoring the counter). Note that M_2 remains deterministic. Finally we add a new outgoing arc to the halting control state of M_1, and we obviously have $p(0) \preccurlyeq q(0)$ iff C does not halt. □

Below we briefly describe two modifications of this construction which show that M_1 or M_2 can be fixed (not depending on C), even with just one or two control states respectively. Here δ represents both $\delta^=$ and $\delta^>$ (which are thus equal).

M_1 **fixed:** M_1 has only one state, $Q_1 = \{p\}$, M_2 has control states $Q_2 = \{q_1, q_2, \ldots, q_n, q_*\}$. The common alphabet is $\Sigma = \{\mathbf{i}_1, \mathbf{z}_1, \mathbf{d}_1, \mathbf{a}_2\}$ (we merge $\mathbf{i}_2, \mathbf{z}_2, \mathbf{d}_2$ into one symbol $\mathbf{a}_2$). The transition function in q_i depends on the type of the i-th command of C. δ_1 gives $\emptyset$ and δ_2 gives $\{(q_*, 0)\}$ in all cases not mentioned in the table.

Commands of C	**Transitions of** M_1	**Transitions of** M_2
i: $c_1 := c_1 + 1;$ *goto* j	$\delta_1(p, \mathbf{i}_1) = \{(p, 1)\}$	$\delta_2(q_i, \mathbf{i}_1) = \{(q_j, 0)\}$
i: *if* $c_1 = 0$ *then goto* j *else* $c_1 := c_1 - 1;$ *goto* k	$\delta_1^=(p, \mathbf{z}_1) = \{(p, 0)\}$ $\delta_1^>(p, \mathbf{d}_1) = \{(p, -1)\}$	$\delta_2(q_i, \mathbf{z}_1) = \{(q_j, 0)\}$ $\delta_2(q_i, \mathbf{d}_1) = \{(q_k, 0)\}$
i: $c_2 := c_2 + 1;$ *goto* j	$\delta_1(p, \mathbf{a}_2) = \{(p, 0)\}$	$\delta_2(q_i, \mathbf{a}_2) = \{(q_j, 1)\}$
i: *if* $c_2 = 0$ *then goto* j *else* $c_2 := c_2 - 1;$ *goto* k		$\delta_2^=(q_i, \mathbf{a}_2) = \{(q_j, 0)\}$ $\delta_2^>(q_i, \mathbf{a}_2) = \{(q_k, -1)\}$
i: *halt*		$\delta_2(q_i, \alpha) = \emptyset \quad (\forall \alpha \in \Sigma)$
		$\delta_2(q_*, \alpha) = \{(q_*, 0)\} \quad (\forall \alpha \in \Sigma)$

M_2 **fixed:** M_1 has control states $Q_1 = \{p_1, p_2, \ldots, p_n\}$, M_2 has control states $Q_2 = \{q_0, q_*\}$. The common alphabet is $\Sigma = \{\mathbf{a}_1, \mathbf{i}_2, \mathbf{z}_2, \mathbf{d}_2, \mathbf{h}\}$ (we merge $\mathbf{i}_1, \mathbf{z}_1, \mathbf{d}_1$ into one symbol $\mathbf{a}_1$, and added a new symbol $\mathbf{h}$). δ_1 gives $\emptyset$ in all cases not mentioned in the table.

Theorem 2. *The problem if $p(m) \simeq q(n)$, where $p(m)$ and $q(n)$ are states of two (nondeterministic) one-counter machines, is undecidable.*

Commands of C	**Transitions of M_1**	**Transitions of M_2**
$i\colon c_1 := c_1 + 1;\ \mathit{goto}\ j$	$\delta_1(p_i, \mathbf{a}_1) = \{(p_j, 1)\}$	$\delta_2(q_0, \mathbf{a}_1) = \{(q_0, 0)\}$
$i\colon \mathit{if}\ c_1 = 0\ \mathit{then}\ \mathit{goto}\ j$ $\mathit{else}\ c_1 := c_1 - 1;\ \mathit{goto}\ k$	$\delta_1^{=}(p_i, \mathbf{a}_1) = \{(p_j, 0)\}$ $\delta_1^{>}(p_i, \mathbf{a}_1) = \{(p_k, -1)\}$	
$i\colon c_2 := c_2 + 1;\ \mathit{goto}\ j$	$\delta_1(p_i, \mathbf{i}_2) = \{(p_j, 0)\}$	$\delta_2(q_0, \mathbf{i}_2) = \{(q_0, 1)\}$
$i\colon \mathit{if}\ c_2 = 0\ \mathit{then}\ \mathit{goto}\ j$ $\mathit{else}\ c_2 := c_2 - 1;\ \mathit{goto}\ k$	$\delta_1(p_i, \mathbf{z}_2) = \{(p_j, 0)\}$ $\delta_1(p_i, \mathbf{d}_2) = \{(p_k, 0)\}$	$\delta_2^{=}(q_0, \mathbf{z}_2) = \{(q_0, 0)\}$ $\delta_2^{>}(q_0, \mathbf{z}_2) = \{(q_*, 0)\}$ $\delta_2^{>}(q_0, \mathbf{d}_2) = \{(q_0, -1)\}$ $\delta_2^{=}(q_0, \mathbf{d}_2) = \{(q_*, 0)\}$
$i\colon \mathit{halt}$	$\delta_1(p_i, \mathbf{h}) = \{(p_i, 0)\}$	$\delta_2(q_0, \mathbf{h}) = \emptyset$
		$\delta_2(q_*, \alpha) = \{(q_*, 0)\} \quad (\forall \alpha \in \Sigma)$

Proof. For control states p and q, taken from M_1 and M_2 respectively, we give a construction of M_1' with r_1 and M_2' with r_2 so that $p(0) \preccurlyeq q(0)$ iff $r_1(0) \simeq r_2(0)$; recalling the previous theorem we shall be done.

We take M_1' to be the disjoint union of M_1 and M_2, adding a new control state r_1 and putting $\delta_1'(r, a) = \{(p, 0), (q, 0)\}$ (a is an arbitrary symbol; here is the only use of nondeterminism when M_1 and M_2 are deterministic). M_2' arises from M_2 by adding a new control state r_2 and putting $\delta_2'(r_2, a) = \{(q, 0)\}$.

It is easily seen that $r_2(0) \preccurlyeq r_1(0)$, and that $r_1(0) \preccurlyeq r_2(0)$ iff $p(0) \preccurlyeq q(0)$. □

Remark. Decidability of simulation equivalence for deterministic one-counter machines follows from decidability of equality for one-counter languages [6]. Undecidability of simulation preorder is slightly stronger than the well-known fact that inclusion for deterministic context-free languages is undecidable.

4 Proof of the Belt Theorem

By an ***area*** we mean a set $A \subseteq \mathbb{N} \times \mathbb{N}$. We define its ***interior*** and ***border*** as follows.

$$\begin{aligned} \mathbf{interior}(A) &= \Big\{ \langle m, n \rangle \ : \ \{m-1, m, m+1\} \times \{n-1, n, n+1\} \ \subseteq \ A \Big\}; \\ \mathbf{border}(A) &= A - \mathbf{interior}(A). \end{aligned}$$

Given an area A and a vector $v \in \mathbb{Z} \times \mathbb{Z}$ (where $\mathbb{Z}$ denotes the set of integers), we let $\mathbf{shift}(A, v) = (A + v) \cap (\mathbb{N} \times \mathbb{N})$ denote the area A shifted by vector v. We say that the shift of an area A by a vector v is ***safe*** wrt $B \subseteq \mathbf{shift}(A, v)$ iff for all graphs $\mathbb{G}_{\langle p,q \rangle}$ and all $u \in B$ we have that $\mathbb{G}_{\langle p,q \rangle}(u)$ is black whenever $\mathbb{G}_{\langle p,q \rangle}(u - v)$ is black. We say that such a shift is ***safe*** iff it is safe wrt $\mathbf{shift}(A, v)$, that is, if it never shifts a black point to a white point.

By a ***line*** ℓ we mean a line with a finite rational slope $\beta > 0$; however, we occasionally refer explicitly to horizontal or vertical lines. We also view a line as a function, writing $\ell(y)$ to represent the value x such that the point $\langle x, y \rangle$ is on the line. We often refer to areas determined by a horizontal line at level $b \in \mathbb{N}$ and one or two lines. For this, we use the following notation: $A[b, \overrightarrow{\ell}, \overleftarrow{\ell'}\,]$ denotes the set of all points of $\mathbb{N} \times \mathbb{N}$ which lie on or above level b, on or to the right of ℓ, and on or to the left of ℓ'. We omit b when $b = 0$. Finally, by a ***belt*** we mean the set of points on or between two parallel lines; here we also allow horizontal and vertical lines. Thus we may have a horizontal belt, or a vertical belt, or a belt of the form $A[\overrightarrow{\ell}, \overleftarrow{\ell'}]$ where ℓ and ℓ' are parallel lines with ℓ' to the right of ℓ.

We can partition the frontiers according to whether or not they lie in a horizontal or a vertical belt. To this end we make the following definitions.

(i) **HF** is the set of frontiers f such that $f(n) = \infty$ for some $n \in \mathbb{N}$. We let $\textsc{hb} \in \mathbb{N}$ (the "horizontal bound") be the least value such that $f(\textsc{hb}) = \infty$ for all $f \in \mathbf{HF}$. The frontiers of **HF** are those which lie in a horizontal belt.

(ii) **VF** is the set of frontiers f such that $\lim_{n\to\infty} f(n) < \infty$. We let $\textsc{vb} \in \mathbb{N}$ (the "vertical bound") be the least value such that $f(n) < \textsc{vb}$ for all $f \in \mathbf{VF}$ and all $n \in \mathbb{N}$. The frontiers of **VF** are those which lie in a vertical belt.

(iii) **IF** is the set of interior frontiers, those not appearing in **HF** nor in **VF**.

We now formalize the notion of a line *separating* frontiers. For this, we need the following notions. We refer to a (horizontal) shift of a line ℓ by an amount $i \in \mathbb{Z}$ by $\mathbf{shift}(\ell, i)$; this is the line ℓ' such that $\ell'(y) = \ell(y) + i$. Given $\beta > 0$, we let $\mathbf{step}(\beta) \in \mathbb{N}$ be the least integral horizontal distance which two lines with slope β must be separated so as to fit a unit square between them; this ensures that, given two such lines ℓ and $\ell' = \mathbf{shift}(\ell, \mathbf{step}(\beta))$ we have $A[\overleftarrow{\ell}] \cap \mathbf{interior}(\mathbb{N} \times \mathbb{N}) \subseteq \mathbf{interior}(A[\overleftarrow{\ell'}])$. Note that $\mathbf{step}(\alpha) \leq \mathbf{step}(\beta)$ whenever $\alpha \geq \beta$.

Definition 1. *A line ℓ with rational slope $\beta > 0$* ***separates frontiers above level*** *$b \in \mathbb{N}$ iff:*

(i) *for all $f \in \mathbf{HF}$, $f(b) = \infty$; that is, $b \geq \textsc{hb}$;*
(ii) *for all f, if $f(b) = -1$ then $f(n) = -1$ for all n;*
(iii) *for all $f \in \mathbf{IF}$, $f(b) > \textsc{vb}$;*
(iv) *for all f, if $f(b) \leq \ell(b)$ then $f(n) < \ell(n) - \mathbf{step}(\beta)$ for all $n \geq b$ (in which case we call f an ℓ-**left frontier**);*
(v) *for all f, if $f(b) \geq \ell(b)$ then $f(n) > \ell(n) + \mathbf{step}(\beta)$ for all $n \geq b$ (in which case we call f an ℓ-**right frontier**).*

Thus the ℓ-left and ℓ-right frontiers are separated by a belt with (horizontal) width $2 \cdot \mathbf{step}(\beta)$ centered on the line ℓ. We say simply that a line ***separates frontiers*** *if it separates frontiers above some level.*

The next Lemma shows that there always exists such a separating line.

Lemma 1. *There is a line ℓ (with rational slope $\beta > 0$) which separates frontiers, in which the ℓ-right frontiers are exactly those of* **HF**.

We now outline the proof of our Belt Theorem.

Proof of The Belt Theorem: Suppose we have a line ℓ with rational slope β which separates frontiers above level b in such a way that ℓ-right frontiers lie in belts and their number cannot be increased by choosing a different ℓ. That such a separating line exists is ensured by Lemma 1.

Let $\mathcal{L}$ be the set of ℓ-left frontiers, and suppose for the sake of contradiction that $\mathcal{L} - \textsf{\textbf{VF}} \neq \emptyset$ (otherwise we have nothing to prove).

For any $n \geq b$, let $\mathbf{gap_1}(n)$ be the (horizontal) distance from ℓ to the rightmost ℓ-left frontier point on level n; that is, $\mathbf{gap_1}(n) = \min\{\ell(n) - f(n) : f \in \mathcal{L}\}$. Note that, since β is rational, the fractional part of $\mathbf{gap_1}(n)$ ranges over a finite set. Hence we cannot have an infinite sequence of levels $i_1, i_2, i_3, \ldots$ above b such that $\mathbf{gap_1}(i_1) > \mathbf{gap_1}(i_2) > \mathbf{gap_1}(i_3) > \cdots$. We can thus take an infinite sequence $i_1 < i_2 < i_3 < \cdots$ of levels above b such that

1. $\mathbf{gap_1}(i) \leq \mathbf{gap_1}(n)$ for all $i \in \{i_1, i_2, i_3, \ldots\}$ and all $n \geq i$;
2. either $\mathbf{gap_1}(i_1) = \mathbf{gap_1}(i_2) = \mathbf{gap_1}(i_3) = \cdots$
   or $\mathbf{gap_1}(i_1) < \mathbf{gap_1}(i_2) < \mathbf{gap_1}(i_3) < \cdots$;
3. for some fixed ℓ-left frontier $f_{\max} \in \mathcal{L}$: $\mathbf{gap_1}(i) = \ell(i) - f_{\max}(i)$ for all $i \in \{i_1, i_2, i_3, \ldots\}$.

The above conditions can be satisfied by starting with the infinite sequence $b+1, b+2, b+3, \ldots$, and first extracting an infinite subsequence which satisfies the first condition, then extracting from this a further infinite subsequence which satisfies (also) the second condition, and then extracting from this a further infinite subsequence which satisfies (also) the third condition.

For $i \in \{i_1, i_2, i_3, \ldots\}$, we let $\mathbf{offset}_i : \mathcal{L} \to \mathbb{N}$ be defined by $\mathbf{offset}_i(f) = f_{\max}(i) - f(i)$. We can then assume that our infinite sequence further satisfies the following condition.

4. For each $f \in \mathcal{L}$: either $\mathbf{offset}_{i_1}(f) = \mathbf{offset}_{i_2}(f) = \mathbf{offset}_{i_3}(f) = \cdots$
   or $\mathbf{offset}_{i_1}(f) < \mathbf{offset}_{i_2}(f) < \mathbf{offset}_{i_3}(f) < \cdots$.
   In the first case, we call f a ***fixed-offset frontier***; and in the second case, we call f an ***increasing-offset frontier***.

This condition can be satisfied by repeatedly extracting an infinite subsequence to satisfy the condition for each $f \in \mathcal{L}$ in turn. Finally, we assume our sequence satisfies the following condition.

5. We have a maximal number of fixed-offset frontiers; no other sequence satisfying conditions 1–4 can have more ℓ-left frontiers $f \in \mathcal{L}$ with $\mathbf{offset}_{i_1}(f) = \mathbf{offset}_{i_2}(f) = \cdots$.

For technical reasons, we also suppose the next two conditions which can be satisfied by dropping some number of initial levels (that is, sequence elements).

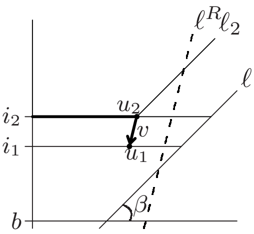
6. $\mathbf{gap_2}(i_1) > |\mathcal{L}| \cdot \mathbf{step}(\beta)$, where $\mathbf{gap_2}(i_j)$ is defined as

$$\min\{\mathbf{offset}_{i_j}(f) : f \text{ is an increasing-offset frontier}\} \\ - \max\{\mathbf{offset}_{i_j}(f) : f \text{ is a fixed-offset frontier}\}$$

7. $f_{\max}(i_1) < f_{\max}(i_2)$.

The line going through the points $u_1 = \langle f_{\max}(i_1), i_1\rangle$ and $u_2 = \langle f_{\max}(i_2), i_2\rangle$ has some slope $\alpha \geq \beta$. If we let **left-of**(u_2) denote the set of points consisting of u_2 along with all points to its left (that is, all (m, i_2) with $m \leq f_{\max}(i_2)$) then the shift of **left-of**(u_2) by $v = u_1 - u_2$ is safe: for the shift of the point onto the y-axis, this is assured by condition (ii) of Definition 1; and for the remaining points, this is assured since frontier offsets cannot shrink (condition 4 above). We can thus invoke the following.

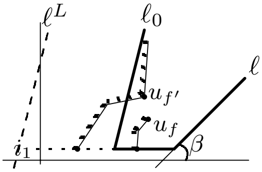
Right Lemma *Consider a line ℓ with slope β separating frontiers above level b, and take two points $u_1 = \langle m_1, i_1\rangle$ and $u_2 = \langle m_2, i_2\rangle$ in $A[b, \overleftarrow{\ell}\,]$ with $m_1 < m_2$ and $b < i_1 < i_2$ such that the slope α of the vector $v = u_1 - u_2$ is at least β. Let ℓ_2 be the line parallel to ℓ which goes through u_2, and suppose that all ℓ-left frontier points in $A[i_2]$ are in $A[\overleftarrow{\ell_2}]$, and that the shift of* **left-of**(u_2) *by v is safe. Then there is a line ℓ^R with slope α separating the same frontiers as ℓ does above level i_2.*



This gives us a line ℓ^R with slope α separating the same frontiers as ℓ above i_2.

Now, there must then be a line ℓ_0 with slope α to the left of ℓ above level i_1 such that every fixed-offset frontier appears in (that is, intersects with) $A = A[i_1, \overrightarrow{\ell_0}, \overleftarrow{\ell}\,]$, and such that whenever a frontier f appears in A there is a frontier point $u_f = \langle f(n), n\rangle \in$ **interior**(A) such that $f(n) - \ell_0(n) \geq f(i_1) - \ell_0(i_1)$; that is, u_f is at least as far to the right of ℓ_0 as the frontier point of f on level i_1. (We can first consider ℓ_0 to be the line going through the frontier points at levels i_1 and i_2 of the fixed-offset frontier with the greatest offset value; if some frontier f is on the border but not in the interior of A, then we can instead take ℓ_0 to be the shift of this line by $-$**step**(α). If there is now some other frontier which is on the border but not in the interior of A, then we again shift the line by $-$**step**(α). We need only shift (at most) once for each frontier in $\mathcal{L}$ before being guaranteed to arrive at a suitable choice for ℓ_0, so we shift by at most $|\mathcal{L}| \cdot$ **step**$(\alpha) \leq |\mathcal{L}| \cdot$ **step**(β), and hence by condition 5 we don't reach the increasing-offset frontiers on level i_1.) We may then invoke the following.

Left Lemma *Suppose we have a line ℓ with rational slope β separating frontiers above level i_1, and a line ℓ_0 with rational slope $\alpha \geq \beta$ to the left of ℓ above level i_1. Suppose further that whenever a frontier f appears in $A = A[i_1, \overrightarrow{\ell_0}, \overleftarrow{\ell}\,]$, there is a frontier point $u_f = \langle f(n), n\rangle \in$* **interior**$(A)$ *such that $f(n) - \ell_0(n) \geq f(i_1) - \ell_0(= i_1)$. Then there is a line ℓ^L to the left of ℓ_0 with slope α such that $f \subseteq A[\overrightarrow{\ell^L}]$ for each such frontier f.*



The premise of this is thus satisfied, so all frontiers with frontier points in A are in $A[\overrightarrow{\ell^L}]$ for some line ℓ^L with slope α. Hence they are in the belt

$A[\overrightarrow{\ell^L}, \overleftarrow{\ell^R}]$ above i_2; and in fact only the fixed-offset frontiers can (and do) have frontier points in $A = A[i_1, \overrightarrow{\ell_0}, \overleftarrow{\ell}\,]$, for otherwise they would not correspond to increasing-offset frontiers.

It remains to demonstrate that we can choose ℓ^L so that it separates frontiers. This can only fail if an increasing-offset frontier appears infinitely often in $A[\,\overrightarrow{\ell'}\,]$ where $\ell' = \mathbf{shift}(\ell^L, -2 \cdot \mathbf{step}(\alpha))$. But then there would be two levels i_{j_1} and i_{j_2} where $f_{\max}(i_{j_1}) - \ell^L(i_{j_1}) = f_{\max}(i_{j_2}) - \ell^L(i_{j_2}) = d$ and $\mathbf{gap_2}(i_{j_1}) > d + |\mathcal{L}| \cdot \mathbf{step}(\alpha)$. We could then find a contradiction using Left Lemma, by considering now the area $A[i_{j_1}, \overrightarrow{\ell^L}, \overleftarrow{\ell^R}]$. □

References

1. Abdulla, P., K. Čerāns (1998). Simulation is decidable for one-counter nets (Extended Abstract.) In Proceedings of CONCUR'98, *Lecture Notes in Computer Science* **1466**:253–268. 405
2. Jančar, P. (1997). Decidability of bisimilarity for one-counter processes. In Proceedings of ICALP'97, *Lecture Notes in Computer Science* **1256**:549–559. (Revised version to appear in *The Journal of Information and Computation*.) 405
3. Jančar, P., F. Moller (1999). Simulation on one-counter nets via colouring. Uppsala Univ. CSD Report No. 159, February 1999. http://www.csd.uu.se/papers/reports.html 405
4. Moller, F. (1996). Infinite results. Proceedings of CONCUR'96, *Lecture Notes in Computer Science* **1119**:195–216. 405
5. Minsky, M. (1967). Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs, NJ. 407
6. Valiant, L.G., M.S. Paterson (1975). Deterministic one-counter automata. *Journal of Computer and System Science* **10**:340–350. 409

On Semantics of Petri Nets over Partial Algebra

Gabriel Juhás*

Fach Informatik, Mathematisch-Geographische Fakultät
Katolische Universität Eichstätt, Ostenstr. 26, 85072 Eichstätt, Germany
Tel.: ++49 8421 931712
gabriel.juhas@ku-eichstaett.de

Abstract. In the paper we study "first consume then produce" and "first produce then consume" causality of transition firings in Petri nets over partial algebra. We show that both causalities may be described by simple algebraic operations on transition systems, namely the compatible and linear operation. Thus, weak semantics (weak marking graph) corresponding to the "first consume then produce" causality is constructed by compatible operation on related net. Strong semantics (strong marking graph), which corresponds to consumption and production in any order, *i.e.*, to combination of both causalities, is given by intersection of compatible and linear image of the related net. Furthermore, it is shown that the classes of weak and strong marking graphs of all Petri nets over arbitrary partial groupoids are equivalent up to isolated markings. The same equivalence is shown for classes of weak and strong marking graphs of all Petri nets over partial groupoids embeddable into Abelian groups.

1 Introduction

In the last decade we can see a substantial effort to develop an abstract and uniform constructions for Petri nets. Main motivation of this effort is to offer a unified theory for different Petri net classes. Most of such abstractions are based on algebraic characterization of Petri nets. A first attempt may be found in a bit forgotten work [6], where marking graphs of p/t nets are shown to be compatible closures of relations over commutative semigroups. A similar approach is given in [10], where the marking graph of a p/t net (understood as directed graph over a commutative monoid) is constructed by symmetric and additive closure of the p/t net. There are many other interesting frameworks, such as Abstract Petri nets [11] working over commutative semigroups, Cone group Petri nets [2,12] working over positive cones of groups, or in particular, over positive cones of Abelian groups [5]. All these frameworks give algebraic characterizations for more or less general class of Petri nets. However they give no answer for Petri net extensions with modified enabling rule and fail even for elementary nets or p/t nets with capacity, but also for nets with inhibitor arcs.

In [8,9] we tried to solve this problem using *partial* groupoids as the Petri net underlying structure. However, for Petri nets over partial groupoids (called

[*] supported by DFG: Project "SPECIMEN"

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 414–422, 1999.

shortly PG nets) causality of transition firings plays an important role. As may be illustrated in example of elementary nets (which may be understood as p/t nets with capacity 1 for every place), considering "first consume then produce" causality one gets that self-loop transitions are enabled to fire when in the related place is a token. In such a case, this token is first consumed and than again produced, so we never exceed capacity of the place. When one considers consumption and production of the token in any order, then self-loop transitions are never enabled to fire. Following terminology introduced for p/t nets with capacity (see e.g. [4,1,3]), we say that semantics (*i.e.* transition system called marking graph) corresponding to the "first consume than produce" causality is *weak*. On the other hand, semantics (marking graph) which corresponds to consumption and production in any order, *i.e.*, to combination of both "first produce then consume" and "first produce then consume" causalities, is called *strong*. One of the aim in mathematical description of Petri nets using partial groupoids is to deal with enabling rule without its explicit definition, using just simple algebraic operations. Thus, in the paper we associate both causalities with such simple operations on transition systems enriched by a partial binary operation on set of states. Then, weak marking graph is constructed by compatible operation on related PG net. Strong marking graph is given by intersection of compatible and linear image of the related PG net. There is a natural question concerning relationships between classes of weak and strong marking graphs (of particular classes) of PG nets. As we show on elementary nets, this classes may differ. However, we show that the classes of weak and strong marking graphs of all Petri nets over arbitrary partial groupoids are equivalent up to isolated markings. Here let us mention an important class of Petri nets over partial groupoids embeddable into Abelian groups (shortly AGE nets). In [9] we have characterized class of weak marking graphs of AGE nets up to isolated markings. The question is whether this characterization works also for strong marking graphs of AGE nets. As we show in this paper, the answer is positive, because classes of weak and strong marking graphs of AGE nets are equivalent up to isolated markings.

2 Basic Definitions

Definition 1. *A* place/transition net *(a* p/t net*) is a quadruple* $\mathcal{N} = (P, T, pre, post)$*, where* P *and* T *are distinct sets of* places *and* transitions*; and* $pre, post\colon T \to P^{\oplus}$ *are* input *and* output *functions attaching multi-sets of places to transitions.*

Given a p/t net $\mathcal{N} = (P, T, pre, post)$, we say that a transition $t \in T$ is *enabled to fire* in a marking $m \in P^{\oplus}$ iff $\exists x \in P^{\oplus} \colon x + pre(t) = m$ and then its *firing* leads to $m' = x + post(t)$. As usual, we write $m \xrightarrow{t} m'$ to denote that t is enabled in m and fires to m'.

The following more general description of the above described enabling and firing rule may be adapted from [6]: Let $(H, +)$ be a commutative semigroup and let $R \subseteq H \times H$. Relation $CR = \{(x+a, x+b) \mid x \in H \wedge (a,b) \in R\}$ is called *compatible image* of R. Now, let $R = \{(pre(t), post(t)) \mid t \in T\}$. Clearly, we have: $(m, m') \in CR \iff \exists t \in T\colon m \xrightarrow{t} m'$. However, using this construction one forgets information about transitions. In order to keep this information, let us extend the notion of compatible image for labeled transition systems.

Definition 2. *A* labeled transition system *(shortly* transition system*) is a tuple* $\mathcal{S} = (S, T, Tran)$ *with set of* states S*, set of* labels L*, and* transition relation $Tran \subseteq S \times L \times S$*.*

Definition 3. *Given a p/t net* $\mathcal{N} = (P, T, pre, post)$*, the transition system with set of states* $S = P^{\oplus}$ *equal to set of all multi-sets over set of places* P*, with set of labels* $L = T$*, and with transition relation* $Tran = \{(m, t, m') \mid m \xrightarrow{t} m'\}$ *is called* (sequential) marking graph *of the net* $\mathcal{N}$*.*

Let us define an *algebraic transition system* to be a transition system with a binary operation $+$ on set of states S. Then, *compatible image* (w.r.t. $(S,+)$) of an algebraic transition system $\mathcal{S} = (S, L, Tran)$ is the transition system $\mathcal{CS} = (S, L, CTran)$ such that $CTran = \{(x+a, t, x+b) \mid x \in S \ \wedge \ (a,t,b) \in Tran\}$[1]. Now, when we understand a p/t net $\mathcal{N} = (P, T, pre, post)$ itself to be an algebraic transition system $\mathcal{S}_{\mathcal{N}} = (P^{\oplus}, T, Tran = \{(pre(t), t, post(t)) \mid t \in T\})$ with multiset addition, then we see that $(m, t, m') \in CTran \iff m \xrightarrow{t} m'$. In other words, compatible image $\mathcal{CS}_{\mathcal{N}}$ of the "net" $\mathcal{S}_{\mathcal{N}}$ is in fact (sequential) marking graph of the net $\mathcal{N}$.

Now, let us recall definitions of elementary nets and p/t nets with capacity – well known classes of Petri nets, which cannot be described by previously mentioned constructions over total algebras.

Definition 4. *An* elementary net *is a quadruple* $\mathcal{E} = (P, T, pre, post)$ *with distinct sets of* places *and* transitions *and* input *and* output *functions* $pre, post\colon T \to 2^P$ *associating a subset of places with every transition.*

Given an elementary net $\mathcal{E} = (P, T, pre, post)$, we say that t is *weakly enabled to fire* in $m \subseteq P$ iff $\exists x \subseteq P\colon x \cap pre(t) = \emptyset \wedge x \cup pre(t) = m \ \wedge x \cap post(t) = \emptyset$ and then its firing leads to $m' = x \cup post(t)$.

Thus, weak enabling rule allows to fire transitions with $pre(t) \cap post(t) \neq \emptyset$, *i.e.*, transitions with self-loops, or in other words transitions with side conditions.

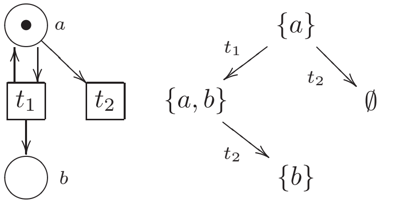
We say that t is *strongly enabled to fire* in $m \subseteq P$ iff $\exists x \subseteq P\colon x \cap pre(t) = \emptyset \wedge x \cup pre(t) = m \ \wedge m \cap post(t) = \emptyset$ and then its *firing* leads to $m' = x \cup post(t)$. Thus, strong enabling rule does not allow firing of transitions with side conditions. In the rest of the paper we suppose definition of the *weak* and *strong marking graph* (case graph) analogously with Definition 3, considering weak and strong enabling rule, respectively. Let us mention, that most of the papers about elementary nets use strong enabling rule. As it is proved in [7], strong marking

[1] One can see $\mathcal{C}$ as a unary operation which attaches to $\mathcal{S}$ its compatible image $\mathcal{CS}$

graphs of each elementary net $\mathcal{E} = (P, T, pre, post)$ preserve so called *diamond property*, *i.e.*, $\forall m, m', m'', m''' \subseteq P \forall\ t_1, t_2 \in T$:

$$\begin{tikzcd}
& m \arrow[ld, "t_1"'] \arrow[rd, "t_2"] & & & & \\
m' \arrow[rd, "t_2"'] & & m'' & \Longrightarrow & & m'' \arrow[ld, "t_1"'] \\
& m''' & & & m''' &
\end{tikzcd}$$

As following example with an elementary net and its weak marking graph shows, it is no longer true for weak marking graphs of elementary nets.



Definition 5. *A* p/t net with capacity $\mathcal{KN}$ *is a p/t net* $\mathcal{N} = (P, T, pre, post)$ *with a* capacity function $K \colon P \to \mathbb{N} \cup \{\infty\}$.

Given a p/t net $\mathcal{N} = (P, T, pre, post)$ with a capacity K, we say that a transition t is *weakly enabled to fire* in $m \in P^{\oplus}$ such that $\forall p \in P \colon m(p) \leq K(p)$ iff $\exists x \in P^{\oplus} \colon x + pre(t) = m \wedge \forall p \in P \colon x + post(t) \leq K(p)$ and then its *firing* leads to $m' = x + post(t)$.

A transition t is *strongly enabled to fire* in $m \in P^{\oplus}$ such that $\forall p \in P \colon m(p) \leq K(p)$ iff $\exists x \in P^{\oplus} \colon x + pre(t) = m \wedge \forall p \in P \colon m + post(t) \leq K(p)$ and then its *firing* leads to $m' = x + post(t)$.

Clearly, one can see elementary nets as p/t nets with constant capacity $K(p) = 1$ for every $p \in P$. Conceptually, weak enabling rule corresponds to the first consumption of number of tokens given by $pre(t)$ and then production of the number of tokens given by $post(t)$. So, we say that it corresponds to the "first consume then produce" causality of transition firings. Strong enabling rule corresponds to the consumption and production of tokens in any order, in other words to the combination of both "first consume then produce" and "first produce then consume" causalities of transition firings [4].

There is a natural question – how to describe elementary nets, p/t nets with capacity, or generally p/t nets with a specific enabling rule in similar way to our extension of Hack's approach of compatible closure, *i.e.*, without explicit definition of enabling rule.

3 Petri Nets over Partial Algebra

In [8,9] we have defined Petri nets over partial groupoids as follows:

Definition 6. *A* Petri net over a partial groupoid *(shortly a* PG net*) is a quadruple* $\mathcal{N} = ((H, dom_+, +), T, pre, post)$ *where* $(H, dom_+, +)$ *is a partial groupoid,* i.e.*,* H *is an arbitrary set and* $+\colon dom_+ \to H$ *is a partial operation defined on domain* $dom_+ \subseteq H \times H$*; and* $pre, post\colon T \to H$ *are* input *and* output *functions, respectively.*

Here let us note that one could easily use concept of places, with a partial groupoid attached to every place, for PG nets. Then, partial groupoid $(H, dom_+, +)$ would be constructed as direct product of these partial groupoids.

Given a PG net $\mathcal{N} = ((H, dom_+, +), T, pre, post)$ we say that a transition t is *enabled to fire* in a marking (a state) $m \in H$ iff $\exists x \in H\colon (x, pre(t)) \in dom_+ \wedge x + pre(t) = m \ \wedge \ (x, post(t)) \in dom_+$; and then its *firing* leads to $m' = x + post(t)$. As usual we write $m \xrightarrow{t} m'$ to denote that t is enabled to fire in m and it leads to m'. Also, we suppose the definition of the *marking graph* of a PG net in similar way as in Definition 3. This enabling/firing rule corresponds to the causality "first consume then produce" and therefore will be called *weak enabling/firing rule* and corresponding marking graph will be called *weak*.

Clearly, we can straightforwardly extend our definition of algebraic transition systems to *partial algebraic transition systems* considering partial groupoid $(S, dom_+, +)$ instead of total groupoid $(S, +)$ on set of states. *Compatible image* (w.r.t. $(S, dom_+, +)$) *of a partial algebraic transition system* $\mathcal{S} = (S, L, Tran)$ is a transition system $\mathcal{CS} = (S, L, CTran)$ such that $CTran = \{(x + a, t, x + b) \mid x \in S \ \wedge \ (a, t, b) \in Tran \wedge (x, a), (x, b) \in dom_+\}$. When we understand a PG net $\mathcal{N} = ((H, dom_+, +), T, pre, post)$ itself to be a partial algebraic transition system $\mathcal{S}_\mathcal{N} = (H, T, Tran = \{(pre(t), t, post(t)) \mid t \in T\})$, then we see that $(m, t, m') \in CTran \iff m \xrightarrow{t} m'$, because we have $CTran = \{(x + pre(t), t, x + post(t)) \mid x \in S \wedge t \in T \wedge (x, pre(t)), (x, post(t)) \in dom_+\}$. In other words, compatible image (w.r.t. $(H, dom_+, +)$) $\mathcal{CS}_\mathcal{N}$ of the "net" $\mathcal{S}_\mathcal{N}$ is, in fact, weak marking graph of the net $\mathcal{N}$.

To generalize "strong enabling/firing rule" let us first formalize the causality "first produce then consume" (shortly PC causality). Using this causality we first produce temporary marking $y = m + post(t)$ and then we consume, *i.e.*, we go to the marking m' which is the rest after we consume $pre(t)$ from y, *i.e.*, we go to m' such that $m' + pre(t) = y = m + post(t)$. Coming back to the nets over partial algebra we can write the PC causality as the following *PC condition*: $(m, post), (m', pre) \in dom_+ \ \wedge \ m + post(t) = m' + pre(t)$. Let us remark that formally one could define "PC enabling/firing rule", but it is not used in any kind of Petri nets (however it is discussed for p/t nets with capacity in [4]).

Thus, strong enabling rule (corresponding to both causalities) will say that a transition $t \in T$ is *strongly enabled to fire* in $m \in H$ iff $\exists x \in H$ such that $(x, pre(t)) \in dom_+ \wedge x + pre(t) = m \ \wedge \ (x, post(t)) \in dom_+$ and PC condition holds for $m' = x + post(t)$; and then *firing* of t leads to m'. We write $m \overset{t}{\rightarrowtail} m'$ to denote that t is strongly enabled in m and its firing leads to m'.

In diagrams (using $a \overset{b}{\longmapsto} c$ to denote that $(a, b) \in dom_+$ and $a + b = c$) we can express weak enabling/firing rule by the left, PC condition by the middle, and strong enabling/firing rule by the commuting right diagram:

$$\begin{tikzcd}
m \arrow[rr, "t"] & & m' \\
& x \arrow[ul, mapsto, "pre(t)"] \arrow[ur, mapsto, "post(t)"'] &
\end{tikzcd}$$

$$\begin{tikzcd}
& y & \\
m \arrow[ur, mapsto, "post(t)"] & & m' \arrow[ul, mapsto, "pre(t)"']
\end{tikzcd}$$

$$\begin{tikzcd}
& y & \\
m \arrow[ur, mapsto, "post(t)"] \arrow[rr, tail, "t"] & & m' \arrow[ul, mapsto, "pre(t)"'] \\
& x \arrow[ul, mapsto, "pre(t)"] \arrow[ur, mapsto, "post(t)"'] &
\end{tikzcd}$$

Clearly, for total commutative semigroups weak and strong enabling rules coincide.

Now there is a question: Is there a "sympathetic" mathematical operation which corresponds to PC condition (and therefore enables to construct strong enabling/firing rule)? Let $(H, +)$ be a commutative semigroup and let $R \subseteq H \times H$. Set of all solutions $x, y \in H$ for equations $x + b = y + a$, where $(a, b) \in R$ is called linear image of R. More precisely, the *linear image* of R is relation $LR = \{(x, y) \mid x, y \in H \wedge (a, b) \in R \wedge x + b = y + a\}$. Thus, for better illustration, considering real numbers with multiplication $(\mathbb{R}, \cdot)$ and a pair of numbers $(a, b) \in \mathbb{R} \times \mathbb{R}$, the linear image of this pair of numbers (or more precisely linear image of one element relation $R = \{(a, b)\}$ is the line $y = x \cdot a/b$.

Linear image (w.r.t. $(S, dom_+, +)$) *of a partial algebraic transition system* $\mathcal{S} = (S, L, Tran)$ is the transition system $\mathcal{LS} = (S, L, LTran)$ such that $LTran = \{(x, t, y) \mid x, y \in S \wedge (a, t, b) \in Tran \wedge (x, b), (y, a) \in dom_+ \wedge x + b = y + a\}$. When we again understand a PG net $\mathcal{N} = ((H, dom_+, +), T, pre, post)$ to be itself a partial algebraic transition system $\mathcal{S}_{\mathcal{N}} = (H, T, Tran = \{(pre(t), t, post(t)) \mid t \in T\})$ then we see that $(m, t, m') \in (CTran \cap LTran) \iff m \overset{t}{\rightarrowtail} m'$.

Let us define *intersection* of two transition systems $\mathcal{S} = (S, L, Tran)$ and $\mathcal{S}' = (S', L', Tran')$ to be transition system $\mathcal{S} \cap \mathcal{S}' = (S \cap S', L \cap L', Tran \cap Tran')$. Now, we can conclude that strong marking graph of a PG net $\mathcal{S}_{\mathcal{N}}$ is intersection $\mathcal{CS}_{\mathcal{N}} \cap \mathcal{LS}_{\mathcal{N}}$ of its compatible image $\mathcal{CS}_{\mathcal{N}}$ and linear image $\mathcal{LS}_{\mathcal{N}}$.

Thus, elementary nets work over partial groupoid $(2^P, dom_\uplus, \uplus)$ with partial operation $\uplus = \cup|_{dom_\uplus}$ working on domain $dom_\uplus = \{(A, B) \mid A \cap B = \emptyset\}$, while p/t nets with capacity work over partial groupoid $(P^\oplus, dom_{\dot{+}}, \dot{+})$ with $dom_{\dot{+}} = \{(x, y) \mid \forall p \in P \colon x(p) + y(p) \in \{0, \cdots, K(p)\}\}$ and $\dot{+} = +|_{dom_{\dot{+}}}$, where operator $+$ denotes standard addition in $P^\oplus$.

4 Relationships between Classes of Weak and Strong Marking Graphs

As we have shown on the class of elementary nets, given a particular class of partial groupoids the classes of weak and strong marking graphs of all PG nets working over partial groupoids from this class may substantially differ. Now, there is a natural question about relationship between classes of weak and strong marking graphs of all PG over working over arbitrary partial groupoids. Before answering this question, let us define that two transition systems are equivalent iff they differ (up to isomorphism) only in some isolated states and unused labels, *i.e.*, iff they differ only in some of their states and labels which never appear in their transition relations. Formally we have:

Definition 7. *Given any pair of transition systems* $\mathcal{S} = (S, L, \mathit{Tran})$ *and* $\mathcal{S}' = (S', L', \mathit{Tran}')$ *we say that* $\mathcal{S}$ *and* $\mathcal{S}'$ *are* (transitionally) equivalent *iff there exists:*
$S_1 \subseteq S, L_1 \subseteq L\colon \mathit{Tran} \subseteq S_1 \times L_1 \times S_1$
$S_1' \subseteq S', L_1' \subseteq L'\colon \mathit{Tran}' \subseteq S_1' \times L_1' \times S_1'$
such that systems $(S_1, L_1, \mathit{Tran})$ *and* $(S_1', L_1', \mathit{Tran}')$ *are* isomorphic *(*i.e. *there exist bijections* $\alpha\colon S_1 \to S_1'$ *and* $\beta\colon L_1 \to L_1'$ *such that* $\forall (s, a, s') \in S_1 \times L_1 \times S_1\colon (s, a, s') \in \mathit{Tran} \iff (\alpha(s), \beta(a), \alpha(s')) \in \mathit{Tran}'$.

Lemma 1. *Every transition system* $\mathcal{S} = (S, L, \mathit{Tran})$ *is (transitionally) equivalent with the weak marking graph of a PG net.*

Proof. (Sketch) It suffices to take the net $((H, \mathit{dom}_+, +), T = L, \mathit{pre}, \mathit{post})$ such that $H = \mathit{Tran} \cup (L \times \{1, 2\}) \cup S$, $\mathit{dom}_+ = \{((s_1, a, s_2), (a, i)) \mid (s_1, a, s_2) \in \mathit{Tran} \wedge i \in \{1, 2\}\}$, for every $((s_1, a, s_2), (a, i)) \in \mathit{dom}_+\colon (s_1, a, s_2) + (a, i) = s_i$, and for every $a \in L\colon \mathit{pre}(a) = (a, 1) \wedge \mathit{post}(a) = (a, 2)$.

Directly from the fact that the strong marking graph of every PG net is a transition system we have the following consequence.

Corollary 1. *The strong marking graph of every PG net is (transitionally) equivalent with the weak marking graph of a PG net.*

Lemma 2. *The weak marking graph of every PG net* $((H, \mathit{dom}_+, +), T, \mathit{pre}, \mathit{post})$ *is (transitionally) equivalent with the strong marking graph of a PG net* $((H', \mathit{dom}_{\dot{+}}, \dot{+}), T, \mathit{pre}', \mathit{post}')$.

Proof. (Sketch) It suffices to take: $H' = (H \times \{-1, 0, 1\}) \cup \{*\}$, where $* \notin H \times \{-1, 0, 1\}$; $\mathit{dom}_{\dot{+}} = \{((a, -1), (b, 1)) \mid (a, b) \in \mathit{dom}_+\} \cup ((H \times \{0\}) \times (H \times \{1\}))$; for every $((a, -1), (b, 1)) \in \mathit{dom}_{\dot{+}}\colon (a, -1) \dot{+} (b, 1) = (a + b, 0)$, for every $((a, 0), (b, 1)) \in \mathit{dom}_{\dot{+}}\colon (a, 0) \dot{+} (b, 1) = *$; and for every $t \in T\colon \mathit{pre}'(t) = (\mathit{pre}(t), 1) \wedge \mathit{post}'(t) = (\mathit{post}(t), 1)$.

Now, let us focus on *Petri nets over partial groupoids embeddable into Abelian groups*[2], shortly called *AGE nets*. In [9] we have characterized (up to isolated markings) weak marking graphs of this class of nets. Now, one could ask whether the same characterization works for strong marking graphs of AGE nets. As following lemmas show, the answer is positive.

Lemma 3. *The weak marking graph of every AGE net* $((H, \mathit{dom}_+, +), T, \mathit{pre}, \mathit{post})$ *is (transitionally) equivalent with the strong marking graph of an AGE net* $((H', \mathit{dom}_{\dot{+}}, \dot{+}), T, \mathit{pre}', \mathit{post}')$.

[2] *i.e.* partial groupoids $(H, \mathit{dom}_+, +)$ for which there exists an Abelian group $(G, \circ)$ such that $H \subseteq G \wedge + = \circ|_{\mathit{dom}_+}$

Proof. (Sketch) Denote by $(G, \circ)$ the Abelian group to which $(H, \mathit{dom}_+, +)$ may be embedded. Now, it suffices to take: $H' = H \times \{-1, 0, 1\}$; $\mathit{dom}_{\dotplus} = \{((a, -1), (b, 1)) \mid (a, b) \in \mathit{dom}_+\} \cup ((H \times \{0\}) \times (H \times \{1\}))$; for every $((a, -1), (b, 1)) \in \mathit{dom}_{\dotplus}\colon (a, -1) \dotplus (b, 1) = (a + b, 0)$, for every $((a, 0), (b, 1)) \in \mathit{dom}_{\dotplus}\colon (a, 0) \dotplus (b, 1) = (a \circ b, 1)$; and for every $t \in T\colon \mathit{pre}'(t) = (\mathit{pre}(t), 1) \wedge \mathit{post}'(t) = (\mathit{post}(t), 1)$. Clearly, Abelian group given by the direct product of group $(G, \circ)$ with the group of integers with addition is the embedding of partial groupoid $(H', \mathit{dom}_{\dotplus}, \dotplus)$.

Lemma 4. *The strong marking graph of every AGE net $\mathcal{N} = ((H, \mathit{dom}_+, +), T, \mathit{pre}, \mathit{post})$ is (transitionally) equivalent with the weak marking graph of an AGE net.*

Proof. (Sketch) Let Abelian group $(G, \circ)$ be the embedding of $(H, \mathit{dom}_+, +)$. Clearly, for every $m \overset{t}{\rightarrowtail} m'$ we can write $m' = m + \mathit{post}(t) - \mathit{pre}(t)$. However, as we have proven in [9], every transition system (S, L, Tran) for which there exists an Abelian group $(G, \circ)$ and a function $f\colon L \to G$ such that $\forall (s, a, s') \in \mathit{Tran}\colon s' = s + f(a)$ is (transitionally) equivalent with the weak marking graph of an AGE net.

References

1. L. Bernardinello and F. De Cindio. A Survey of Basic Models and Modular Net Classes. In *Advances in Petri Nets 1992*, LNCS 609, pp. 304–351, 1992. 415
2. P. Braun. Ein allgemeines Petri-netz. Master's thesis, Univ. of Frankfurt, 1992. 414
3. J. Desel and W. Reisig. Place/Transition Petri Nets. In *Lectures on Petri nets I: Basic Models*, LNCS 1491, pp. 123–174, 1998. 415
4. R. Devillers. The Semantics of Capacities in P/T Nets. In *Advances in Petri Nets 1989*, LNCS 424, pp. 128–150, 1990. 415, 417, 418
5. M. Droste and R. M. Shrott. Petri Nets and Automata with Concurrency Relation—an Adjunction. In M. Droste and Y. Gurevich (eds) *Semantics of Programming Languages and Model Theory*, Gordon and Breach Sc. Publ., pp. 69–87, 1993. 414
6. M. Hack. Petri Nets and Commutative Semigroups. Computation Structures Note No. 18, Project MAC, M.I.T., July 1974. 414, 415
7. H. J. Hoogeboom and G. Rozenberg. Diamond properties of elementary net systems. *Fundamenta Informaticae*, XIV:287–300, 1991. 416
8. G. Juhás. The essence of Petri nets and transition systems through Abelian groups. *Electronic Notes in Theoretical Computer Science*, 18, 1998. 414, 417
9. G. Juhás. Reasoning about algebraic generalisation of Petri nets. In *Proc. of 20th International Conference on Theory and Application of Petri Nets*, Williamsburg, VA, USA, Springer, LNCS 1639, pp. 324–343, 1999. 414, 415, 417, 420, 421
10. J. Meseguer and U. Montanari. Petri nets are monoids. *Information and Computation*, 88(2):105–155, October 1990. 414

11. J. Padberg. *Abstract Petri Nets: Uniform Approach and Rule-Based Refinement*. PhD thesis, Technical University of Berlin, 1996. 414
12. P. Tix. One FIFO place realizes zero-testing and Turing machines. *Petri net Newsletter*, 51, 12, 1996. 414

Towards Possibilistic Decision Functions with Minimum-Based Sugeno Integrals*

Ivan Kramosil

Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Prague 8
Fax: ++420 2 85 85 789
kramosil@uivt.cas.cz

Abstract. The aim of this contribution is to develop, following the pattern applied in the theory of statistical decision functions, an analogous mathematical tool for decision making under uncertainty supposing that probabilistic measures are replaced by the possibilistic ones and expected values are defined by Sugeno integrals with the minimum taken as the conjunctive t-seminorm function. The model is illustrated by possibilistic decision functions for the case of sequences of coin tosses.

1 Decision Problem under Uncertainty

We shall follow, below, a formalized model of uncertainty definition, quantification and processing enabling to identify uncertainty with the lack of complete information. This approach if often called the *paradigm of hidden parameters.*

Let Ω denote the space of all possible *elementary states* of the universe of discourse so that everything is completely determined by the actual elementary state $\omega_0 \in \Omega$. Let $S = \Omega/\approx$ be the factor space defined in Ω by an equivalence relation $\approx$.

Decision problem, taken as an individual instance of decision making, consists in choosing, more or less rationally and sophistically and on the ground of some, in general partial, knowledge concerning an investigated system and the world around, one decision d from a nonempty set D of possible decisions. In what follows we accept the basic simplifying assumption that there is defined, for each state $s \in S$ and each decision $d \in D$, a nonnegative real number $\lambda(s,d)$ quantifying the loss or expenses which the subject suffers or must pay when she/he takes the decision d supposing that $s \in S$ is the actual state of the world.

However, as a rule, the actual state s_0 is not known to the subject, the only data being at her/his disposal are the results of some observations and/or experiments giving some information about the investigated system and its environment. Let E denote the nonempty space (perhaps a vector one) of possible values of these empirical data. So, what the subject does is that she/he takes

* This work has been sponsored by the grant No. A 1030803 of the GA AS CR.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 422–430, 1999.

a decision from D on the ground of the known value $x \in E$. We shall limit ourselves to the simple case of deterministic decision functions $\delta\colon E \to D$.

So, let $\sigma\colon \Omega \to S$ be the function ascribing to each $\omega \in \Omega$ the corresponding state of the world $\sigma(\omega) = [\omega]_{\approx} \in S = \Omega/\approx$. The often considered case when S is different from the factor-space $\Omega/\approx$ can be proved to be equivalent, at least for our purposes, with the former one when identifying each $s \in S$ with its inverse image $\{\omega \in \Omega : \sigma(\omega) = s\} \subset \Omega$. Let $X\colon \Omega \to E$ ascribe to each $\omega \in \Omega$ the empirical value $X(\omega)$. Let ω_1, $\omega_2 \in \Omega$ be two elementary states such that that $\sigma(\omega_1) = s_1 \neq s_2 = \sigma(\omega_2)$, but $\lambda(s_1, \mathrm{d}_{\lambda,s_2}) > \lambda(s_1, d_{\lambda,s_1})$ or $\lambda(s_2, d_{\lambda,s_1}) > \lambda(s_2, d_{\lambda,s_2})$ holds, where $d_{\lambda,s_i},\ i = 1$, 2, is (the, an) optimal decision supposing that s_i is the actual state. Let, moreover, $x_1 = X(\omega_1) = x_2 = X(\omega_2)$, hence, $\delta(x_1) = \delta(x_2)$, so that at least in one state of the world (s_1 or s_2) the decision taken by the decision function δ is not (the, an) optimal one. Consequently, it is not too reasonable to expect that there exists, in general, a decision function $\delta_{\lambda,\mathrm{opt}}\colon E \to D$ satisfying the equality $\delta_{\lambda,\mathrm{opt}}(X(\omega)) = d_{\lambda,\sigma(\omega)}$ uniformly for all $\omega \in \Omega$. So, we have to limit our effort to decision functions satisfying some weaker optimality conditions than that of the uniform optimality over the space Ω of all elementary states.

The already classical general solution to the just sketched problem of decision making under uncertainty offers the theory of statistical decision functions, [2,10]. The basic idea is that not the pointwise values of the loss function for particular (elementary or macro) states are considered, but rather their expected (average, mean) values over certain classes of (elementary or macro) states. Sizes of classes of elementary states are defined by a probability measure over a σ-field of subsets of the space Ω and expected (mean, average) values of loss functions are defined by their integrals over the resulting probability space.

Recently, possibilistic measures have been introduced and developed as an alternative tool for uncertainty quantification and processing and Sugeno integrals (defined with respect to a fixed t-norm) play the role of mean values. Cf. [4] or [11] for the origins of possibility measure and [3] as an exhaustive surveyal study dealing with the contemporary state of the domain of research. Let us investigate, in what follows, whether it is possible to develop a mathematical model of decision functions within the framework of possibility measures and Sugeno integrals.

2 Possibilistic Measures and Minimum-Based Sugeno Integral

In order to abbreviate our notation we shall use the the symbols $\vee$, $\bigvee$ ($\wedge$, $\bigwedge$, resp.) to denote the supremum (infimum, resp.) operation on the real line and with respect to the standard linear ordering of real numbers, the standard conventions for $\pm\infty$ and for the empty set being accepted.

Let us begin with the definition of possibilistic measure.

Definition 1. *Let Ω be a nonempty set, let $\mathcal{P}(\Omega) = \{A : A \subset \Omega\}$. Possibilistic measure on Ω is a mapping $\mu \colon \mathcal{P}(\Omega) \to \langle 0,1\rangle$ such that $\mu(\emptyset) = 0$, $\mu(\Omega) = 1$ and $\mu(A \cup B) = \mu(A) \vee \mu(B)$ for each A, $B \subset \Omega$.* □

Definition 2. *A mapping $\pi \colon \Omega \to \langle 0,1\rangle$ is called possibilistic distribution on Ω. The possibilistic measure μ_π induced by the possibilistic distribution π on Ω is defined by $\mu_\pi(A) = \bigvee\{\pi(\omega) : \omega \in A\}$ for each $A \subset \Omega$. Possibilistic measure μ is called distributive, if there exists a possibilistic distribution π on Ω such that $\mu \equiv \mu_\pi$ (here and below, $\mu_1 \equiv \mu_2$ means that $\mu_1(A) = \mu_2(A)$ for all $A \subset \Omega$).* □

The following assertions are almost obvious. (i) If μ is a possibilistic measure on Ω, then then mapping $\pi_\mu \colon \Omega \to \langle 0,1\rangle$ defined by $\pi_\mu(\omega) = \mu(\{\omega\})$ for all $\omega \in \Omega$ is a possibilistic distribution on Ω. (ii) If μ is distributive, then $\mu \equiv \mu_{\pi_\mu}$. (iii) If μ_1, μ_2 are two distributive possibilistic measures on Ω such that $\mu_1 \not\equiv \mu_2$, then $\pi_{\mu_1} \not\equiv \pi_{\mu_2}$. (iv) If π_1, π_2 are two possibilistic distributions on Ω such that $\pi_1 \not\equiv \pi_2$, then $\mu_{\pi_1} \not\equiv \mu_{\pi_2}$. (v) If Ω is finite, then each possibilistic measure on Ω is distributive. (vi) There exists non-distributive possibilistic measures.

Definition 3. *Let μ be a possibilistic measure on a nonempty set Ω, let $f \colon \Omega \to \langle 0,1\rangle$ be a real-valued function. The minimum-based Sugeno integral over Ω of the function f with respect to the possibilistic measure μ is defined by*

$$\oint_\Omega f \, \mathrm{d}\mu = \bigvee\nolimits_{\alpha \in \langle 0,1\rangle} \Big[\alpha \wedge \mu(\{\omega \in \Omega : f(\omega) \geq \alpha\})\Big]. \tag{1}$$

The proof of the following assertion is a matter of technical routine.

Lemma 1. *Let Ω be a nonempty set, let μ be a possibilistic measure on Ω, let $f, g \colon \Omega \to I = \langle 0,1\rangle$ be two functions, let $(f \otimes g)(\omega) = f(\omega) \otimes g(\omega)$ for all $\omega \in \Omega$ and for $\otimes = \cdot$ (product), $\wedge$, $\vee$ and $\oplus$, where $x \oplus y = (x+y) \wedge 1$ for all x, $y \in I$. Let $f \leq g$ mean that $f(\omega) \leq g(\omega)$ holds for each $\omega \in \Omega$. Then*

$$\text{(i)} \quad \text{If } f \leq g, \text{ then} \oint f \, \mathrm{d}\mu \leq \oint g \, \mathrm{d}\mu. \tag{2}$$

$$\text{(ii)} \quad \oint fg \, \mathrm{d}\mu \leq \oint (f \wedge g) \, \mathrm{d}\mu \leq \left(\oint f \, \mathrm{d}\mu\right) \wedge \left(\oint g \, \mathrm{d}\mu\right) \leq$$
$$\left(\oint f \, \mathrm{d}\mu\right) \vee \left(\oint g \, \mathrm{d}\mu\right) = \oint (f \vee g) \, \mathrm{d}\mu \leq \oint (f \oplus g) \, \mathrm{d}\mu.$$

Theorem 1. *Let Ω be a nonempty set, let μ be a distributive possibilistic measure on Ω, let $F \colon \Omega \to I$ be a function. Then*

$$\oint F \, \mathrm{d}\mu = \bigvee\nolimits_{\omega \in \Omega} (F(\omega) \wedge \mu(\{\omega\})). \tag{3}$$

Proof. Take $x \in I$ such that $x < \oint F\,\mathrm{d}\mu =_{\mathrm{df}} \bigvee_{\alpha \in I} \alpha \wedge \mu(\{\omega \in \Omega : F(\omega) \geq \alpha\})$ holds. Then there exists $\alpha \in I$ such that $\alpha \wedge \mu(\{\omega \in \Omega : F(\omega) \geq \alpha\}) \geq x$ holds, consequently, $\mu(\{\omega \in \Omega : F(\omega) \geq \alpha\}) \geq x$ and $\alpha \geq x$ hold as well. So, there exists $\omega_0 \in \Omega$ such that $F(\omega_0) \geq \alpha$ and $\mu(\{\omega_0\}) \geq x$ hold, as μ is supposed to be distributive. It follows that $F(\omega_0) \wedge \mu(\{\omega_0\}) \geq \alpha \wedge x = x$, as $\alpha \geq x$. Hence, $\bigvee_{\omega \in \Omega} F(\omega) \wedge \mu(\{\omega\}) \geq x$ holds. Consequently,

$$\bigvee_{\omega \in \Omega} F(\omega) \wedge \mu(\{\omega\}) \geq \bigvee \left\{ x \in I : x < \oint F\,\mathrm{d}\mu \right\} \oint F\,\mathrm{d}\mu. \tag{4}$$

Let the equality in (4) not hold. Then there exists $\omega_0 \in \Omega$ such that

$$F(\omega_0) \wedge \mu(\{\omega_0\}) > \oint F\,\mathrm{d}\mu = \bigvee_{\alpha \in I} \alpha \wedge \mu(\{\omega \in \Omega : F(\omega) \geq \alpha\}) \tag{5}$$

holds. Setting $\alpha = F(\omega_0)$ we obtain that $\mu(\{\omega \in \Omega : F(\omega) \geq \alpha\}) \geq \mu(\{\omega_0\})$, consequently,

$$\begin{aligned} \oint F\,\mathrm{d}\mu &= \bigvee_{\alpha \in I} \alpha \wedge \mu((\{\omega \in \Omega : F(\omega) \geq \alpha\}) \\ &\geq \bigvee_{\alpha = F(\omega_0)} \alpha \wedge \mu((\{\omega \in \Omega : F(\omega) \geq \alpha\}) \\ &= F(\omega_0) \wedge \mu(\{\omega \in \Omega : F(\omega) \geq F(\omega_0)\}) \\ &\geq F(\omega_0) \wedge \mu(\{\omega_0\}) \end{aligned} \tag{6}$$

but this result contradicts the supposed inequality in (4). □

Our definition of Sugeno integral recalls the original idea according to which this notion has been introduced. In [3] Sugeno integral is defined by an expression which generalizes the right-hand side of (3) in the sense that possibilistic measures and integrated functions in question take values in a complete partially ordered lattice Ł and the values $F(\omega)$ and $\mu(\{\omega\})$ are combined together not using the corresponding minimum operation in Ł, but by a binary function P possessing the properties of t-seminorm (cf., again, [3] for more detail). Our definition of Sugeno integral follows as a particular case of the definition from [3] with the adjective "minimum-based" abbreviating the complete specification "minimum-based real-valued Sugeno integral taking its values in the unit interval I of real numbers".

Under the interpretation of possibilistic measure values like complements to an appropriate numerical quantification of the size of the set of arguments putting in doubts the occurrence of the event the possibility measure of which is to be defined, the minimum operation results as a quite natural combinatorial rule for the corresponding particular possibilistic measure values. According to this interpretation, $\mu(A) = 1$, if there are no arguments (known to the subject, say) weakening the subject's belief that (the occurrence of the event) A is possible, and every supplementary information may only make this value smaller (or leave it untouched). Cf. [9] for a more detailed discussion concerning this case.

Moreover, when considering a nonstandard interpretation of real numbers from I (taking them as sequences of binary or ternary digits rather than as numbers) together with their corresponding (partial) ordering, then the nonstandard minimum operation can be converted into standard product (cf. [8] for more detail).

Equality (3) need not hold for non-distributive possibilistic measures (as a counter-example, take again the binary possibilistic measure μ separating the finite and infinite subsets of an infinite space Ω and take $F\colon \Omega \to \{0,1\}$ such that $\{\omega \in \Omega : F(\omega) = 1\}$ is infinite). The following corollary of Theorem 1 easily follows.

Corollary 1. *Let $\{\Omega_1, \Omega_2, \dots, \Omega_n\}$ be a finite partition of the basic space Ω, let $F\colon \Omega \to I$ be such that $F(\omega) = c_i$ for each $\omega \in \Omega_i$, let μ be a distributive possibilistic measure on Ω. Then*

$$\oint F \,\mathrm{d}\mu = \textstyle\bigvee_{i=1}^{n} (c_i \wedge \mu(\Omega_i)). \tag{7}$$

3 Possibilistic Decision Functions

Let Ω be a nonempty set, let μ be a distributive possibilistic measure on Ω. Let S be a nonempty space of (macro) states, let $\sigma\colon \Omega \to S$ be a mapping. Let D be a nonempty set of decisions, let $\lambda\colon S \times D \to \langle 0,1\rangle$ be the loss function, i. e., $\lambda(s,d) \in I = \langle 0,1\rangle$ is the loss suffered (the expenses to be paid) when decision d is taken and s is the actual (macro) state of the world. Let E be the space of possible empirical values, let $\delta\colon E \to D$ be a decision function. Let there exist, for each $s \in S$, a mapping $X_s\colon \Omega \to E$ such that $X_s(\omega) \in E$ is the empirical value being at the subject's disposal when $\omega \in \Omega$ is the actual elementary state of the Universe and $s = \sigma(\omega) \in S$ is the actual state of the world. Hence, $\lambda(s, \delta(X_s(\omega))) \in I$ is the loss suffered when s is the actual state of the world, $X_s(\omega)$ is the empirical value being at the subject's disposal and δ is the used decision function.

The *possibilistic risk (p-risk)* $\rho^p(s,\delta)$ will be defined by the "expected" (in the sense of Sugeno integral) loss resulting when s is the actual state of the world and the decision function δ is used. So, using Theorem 1, and the assumption that μ is distributive, we obtain that

$$\begin{aligned}
\rho^p(s,\delta) &= \oint \lambda(s, \delta(X_s(\omega)))\,\mathrm{d}\mu \\
&= \textstyle\bigvee_{\alpha\in I} \Big[\alpha \wedge \mu(\{\omega \in \Omega : \lambda(s,\delta(X_s(\omega))) \geq \alpha\})\Big] \\
&= \textstyle\bigvee_{\omega\in\Omega} \Big[\lambda(s,\delta(X_s(\omega))) \wedge \mu(\{\omega\})\Big] \\
&= \textstyle\bigvee_{x\in E} \Big[\lambda(s,\delta(x)) \wedge \mu(\{\omega \in \Omega : X_s(\omega) = x\})\Big].
\end{aligned} \tag{8}$$

The reader should always keep in mind that the term "possibilistic risk" introduced above reflects the high degree of *formal* analogy and similarity between the notion of possibilistic risk and that of classical probabilistic risk, leaving aside the question what kind and degree of quality of the possibilistic decision function in question is evaluated and classified by the possibilistic criterion. This question seems to be still open and worth being solved with a great effort, but it would bring us beyond the strictly limited scope and extent of this contribution.

In the simple example of state detection with $S = D$ and $\lambda(s,d) = 0$, if $s = d$, $\lambda(s,d) = 1$, if $s \neq d$, we obtain easily that

$$\rho^p(s,\delta) = \mu(\{\omega \in \Omega : \delta(X_s(\omega)) \neq s\}). \tag{9}$$

So, $\rho^p(s,\delta_s) = 0$ for constant decision function $\delta_s \equiv s$ for each $x \in E$ hence, up to rather simple cases a uniformly with respect to s optimal decision function does not exist. Let us examine the minimax and the Bayesian approach applied to our case.

A decision function $\delta_{p-mm}\colon E \to D$ is called the *possibilistic minimax* $(p-mm)$ *solution* to the decision problem in question, if the inequality

$$\bigvee_{s\in S}\rho^p(s,\delta_{p-mm}) \leq \bigvee_{s\in S}\rho^p(s,\delta) \tag{10}$$

holds for each decision function $\delta\colon E \to D$. Given $\varepsilon > 0$, a decision function $\delta_{\varepsilon-p-mm}\colon E \to D$ is called the ε*--possibilistic minimax* $(\varepsilon - p - mm)$ *solution* to the decision problem in question, if the inequality

$$\bigvee_{s\in S}\rho^p(s,\delta_{s-p-mm}) - \varepsilon \leq \bigvee_{s\in S}\rho^p(s,\delta) \tag{11}$$

holds for each decision function $\delta\colon E \to D$. In general, $p - mm$ solution need not exists, but $\varepsilon - p - mm$ solution always exists for each $\varepsilon > 0$.

Let us define the *possibilistic Bayesian risk* $\rho^{*p}(\sigma(\cdot),\delta)$ with respect to the apriori function σ and decision function δ by

$$\begin{aligned}\rho^{*p}(\sigma(\cdot),\delta) &= \oint \rho^p(\sigma(\omega),\delta)\,\mathrm{d}\mu \\ &= \bigvee_{\alpha\in I}\Big[\alpha \wedge \mu(\{\omega \in \Omega : \rho^p(\sigma(\omega),\delta) \geq \alpha\})\Big].\end{aligned} \tag{12}$$

An easy calculation yields that, for distributive possibilistic measures,

$$\rho^{*p}(\sigma(\cdot),\delta) = \bigvee_{\langle\omega,\omega'\rangle\in\Omega\times\Omega}\Big[\lambda(\sigma(\omega),\, \delta(X_{\sigma_(\omega)}(\omega')))(\mu\times\mu)\,(\{\langle\omega,\omega'\rangle\})\Big] \tag{13}$$

where $\mu\times\mu$ is the distributive possibilistic measure on $\Omega\times\Omega$ (uniquely) defined by the possibilistic distribution

$$(\mu\times\mu)\,(\{\langle\omega,\omega'\rangle\}) = \mu(\{\omega\}) \wedge \mu(\{\omega'\}) \tag{14}$$

for each ω, $\omega' \in \Omega$, and by

$$(\mu\times\mu)\,(C) = \bigvee_{\langle\omega,\omega'\rangle\in C}(\mu\times\mu)\,(\{\langle\omega,\omega'\rangle\}) \tag{15}$$

for each $C \subset \Omega \times \Omega$. Corollary 1 yields that

$$\rho^{*p}(\sigma(\cdot), \delta) = \bigvee_{s \in S} \Big[\bigvee_{x \in E} \lambda(s, \delta(x)) \wedge \mu(\{\omega' \in \Omega : X_s(\omega') = x\}) \Big] \\ \wedge \mu(\{\omega \in \Omega : \delta(\omega) = s\}). \tag{16}$$

A decision function $\delta_{p-b,\sigma} \colon E \to D$ is called the *possibilistic Bayes* ($p{-}b$) *solution* to the decision problem in question with respect to the apriori function σ, if the inequality

$$\rho^{*p}(\sigma(\cdot), \delta_{p-b,\sigma}) \leq \rho^{*p}(\sigma(\cdot), \delta) \tag{17}$$

holds for each decision function $\delta \colon E \to D$. Given $\varepsilon > 0$, a decision function $\delta_{\varepsilon-p-b,\sigma} \colon E \to D$ is called the *ε-possibilistic Bayes* ($\varepsilon - p - b$) *solution* to the decision problem in question with respect to the apriori function σ, if the inequality

$$\rho^{*p}(\sigma(\cdot), \delta_{\varepsilon-p-b,\sigma}) - \varepsilon \leq \rho^{*p}(\sigma(\cdot), \delta) \tag{18}$$

holds for each decision function $\delta \colon E \to D$. As in the minimax case, $p{-}b$-solution need not exist, in general, but $\varepsilon - p - b$ solution always exists for each $\varepsilon > 0$.

Let us note that if $\mu(\{\omega \in \Omega : \sigma(\omega) = s\})$ is the same for all $s \in S$ then the possibilistic minimax solution and the possibilistic Bayes solution to the decision problem in question are identical. Such a situation occurs, e. g., in the case of total apriori ignorance, when for each $s \in S$ there exists $\omega \in \Omega$ such that $\mu(\{\omega\}) = 1$ and $\sigma(\omega) = s$.

4 An Example – Decision Making Based on Coin Tossing

Let $\langle \Omega, \mathcal{P}(\Omega), \mu \rangle$ be as above, let $E_0 = \{H(\text{ead}), T(\text{ail})\}$, let $E = E_0^n$ be the empirical space of n-tuples resulting from n tosses of the tested coin, let $X_1 = \Omega \to E_0$ be such that

$$\mu(\{\omega \in \Omega : X_1(\omega) = H\}) = p^H, \\ \mu(\{\omega \in \Omega : X_1(\omega) = T\}) = p^T. \tag{19}$$

Let $D = \{\text{HYP, ALT}\}$, where $\text{HYP} = \langle p_1^H, p_1^T \rangle$, $\text{ALT} = \langle p_1^H, p_1^T \rangle$. In what follows, we shall simplify our decision space reducing HYP to p_1^H and ALT to p_2^H.

Definition 4. *Let variables $X_1, X_2, \ldots, X_n$ be mappings taking Ω into a nonempty set E. They are called possibilistically (minimum-based) independent with respect to μ, if for each $F_1, F_2, \ldots, F_n \subset E$ the equality*

$$\mu\left(\bigcap_{i=1}^n \{\omega \in \Omega : X_i(\omega) \in F_i\}\right) = \bigwedge_{i=1}^n \mu(\{\omega \in \Omega : X_i(\omega) \in F_i\}) \tag{20}$$

holds. Variables $X_1, X_2, \ldots, X_n$ are called identically possibilistically distributed, if for each $i, j \leq n$ and each $F \subset E$ the equality

$$\mu(\{\omega \in \Omega : X_i(\omega) \in F\}) = \mu(\{\omega \in \Omega : X_j(\omega) \in F\}) \tag{21}$$

holds. As can be easily seen, if μ is distributive, then (21) holds iff it holds for singletons $F = \{x\},\ x \in E$.

Let $X_1 \colon \Omega \to E_0$ be as above, let $X_1, X_2, \ldots, X_n$ be an n-tuple of possibilistically independent and identically possibilistically distributed variables.

Set $X_n^p(\omega) = (1/n)\,\mathrm{card}\{i \leq n : X_i(\omega) = H\}$, where $p = \langle p^H, p^T \rangle$, suppose that $p_1^H > p_2^H$, and define, for all $\beta \in I$, the decision function $\delta_\beta \colon I \to D$ as follows:

$$\delta_\beta(X_n^p(\omega)) = \text{HYP, if } X_n^p(\omega) \geq \beta, \tag{22}$$

$$\delta_\beta(X_n^p(\omega)) = \text{ALT, if } X_n^p(\omega) < \beta. \tag{23}$$

So, writing ρ^{poss} instead of ρ^p to avoid any misunderstanding we obtain that, for $S = \{p_1, p_2\} = \{\langle p_1^H, p_1^T \rangle, \langle p_2^H, p_2^H \rangle\}$,

$$\rho^{\mathrm{poss}}(p_1, \delta_\beta) = \mu(\{\omega \in \Omega : X_n^{p_1}(\omega) < \beta\}), \tag{24}$$

$$\rho^{\mathrm{poss}}(p_2, \delta_\beta) = \mu(\{\omega \in \Omega : X_n^{p_2}(\omega) \geq \beta\}). \tag{25}$$

For the extremum values $\beta = 0$ and $\beta = 1$ we obtain that

$$\rho^{\mathrm{poss}}(p_1, \delta_0) = \mu(\emptyset) = 0,\ \rho^{\mathrm{poss}}(p_2, \delta_0) = \mu(\Omega) = 1, \tag{26}$$

$$\rho^{\mathrm{poss}}(p_1, \delta_1) = \mu(\{\omega \in \Omega : X_n^{p_1}(\omega) < 1\}) = p_1^T \vee (p_1^H \wedge p_1^T) = p_1^T, \tag{27}$$

$$\rho^{\mathrm{poss}}(p_2, \delta_1) = \mu(\{\omega \in \Omega : X_n^{p_2}(\omega) \geq 1\}) = p_2^H. \tag{28}$$

For $\beta = j/n,\ j = 2, \ldots, n-1$ we obtain that

$$\rho^{\mathrm{poss}}(p_1, \delta_\beta) = \mu(\{\omega \in \Omega : X_n^{p_1}(\omega) < \beta\}) = p_1^T \vee (p_1^H \wedge p_1^T) = p_1^T, \tag{29}$$

$$\rho^{\mathrm{poss}}(p_2, \delta_\beta) = \mu(\{\omega \in \Omega : X_n^{p_2}(\omega) \geq \beta\}) = (p_2^H \wedge p_2^T) \vee p_2^H = p_2^H. \tag{30}$$

For $\beta = 1/n$ the situation is the same as for $\beta = 0$, for other values $\beta \in I$ the situation obviously reduces to one of the values j/n. So,

$$\rho^{\mathrm{poss}}(p_1, \delta_\beta) \vee \rho^{\mathrm{poss}}(p_2, \delta_\beta) = 1 \tag{31}$$

if $\beta = 0$ or $\beta = 1/n$, and

$$\rho^{\mathrm{poss}}(p_1, \delta_\beta) \vee \rho^{\mathrm{poss}}(p_2, \delta_\beta) = p_1^T \vee p_2^H \tag{32}$$

if $\beta = j/n,\ j \geq 2$. So, if $p_1^T \vee p_2^H < 1$, then any $\delta_\beta,\ \beta = j/n,\ j \geq 2$ is the possibilistic minimax solution, if $p_1^T \vee p_2^H = 1$, any δ_β plays this role.

Let $\sigma \colon \Omega \to S$ be an apriori mapping, let us denote

$$\pi_j = \mu(\{\omega \in \Omega : \sigma(\omega) = p_j\}) \tag{33}$$

for both $i = 1$, 2. Then

$$\rho^{*\mathrm{poss}}(\sigma, \delta_\beta) = (\pi_1 \wedge \rho^{\mathrm{poss}}(p_1, \delta_\beta)) \vee (\pi_2 \wedge \rho^{\mathrm{poss}}(p_2, \delta_\beta)). \tag{34}$$

In particular, for $\beta = 0$ and $\beta = 1/n$,

$$\rho^{*\mathrm{poss}}(\sigma, \delta_\beta) = (\pi_1 \wedge 0) \vee (\pi_2 \wedge 1) = \pi_2 \tag{35}$$

for $\beta = j/n,\ j \geq 2$, we obtain that

$$\rho^{*\mathrm{poss}}(\sigma, \delta_\beta) = (\pi_1 \wedge p_1^T) \vee (\pi_2 \wedge p_2^H). \tag{36}$$

So, if the inequality $\pi_2 \leq (\pi_1 \wedge p_1^T) \vee (\pi_2 \wedge p_2^H)$ holds, then δ_0 or $\delta_{1/n}$ is the possibilistic Bayes solution to the decision problem in question, if the inverse inequality holds, then every $\delta_\beta,\ \beta = j/n,\ j \geq 2$, presents such a solution. The independence of the results on the value n may be surprising only at the first sight, as the only what matters, in fact, when considering the sequence $X_1^p(\omega), X_2^p(\omega), \ldots, X_n^p(\omega)$ is, whether it contains only H's, only T's, or whether it is "mixed". The presented example is very simple so that possibilistic decision functions seem to be too rough tool to solve it, but perhaps a more sophisticated case would better illustrate the powers of the tools suggested above. Let us refer the reader also to [1,5,6,7], where similar problems like here are presented and discussed.

References

1. S. Benferhat, C. Sossai: Mergind Uncertain Knowledge Bases in a Possibilistic Logic Framework. In: UAI 98 – Proceedings of the 14th Annual Conference (Gregory F. Cooper and Serafim Morales, Eds.), Morgan Kaufmann Publishers, Inc., San Francisco, 1998, pp. 8–15. 430
2. D. Blackwell, N. A. Girshick: Theory of Games and Statistical Decisions. John Wiley and Sons, New York, 1954. 423
3. G. De Cooman: Possibility Theory – Part I, II, III. Int. J. General Systems 25 (1997), No. 4, pp. 291–323 (Part I), 325–351 (Part II), 353–371 (Part III). 423, 425, 425, 425
4. D. Dubois, H. Prade: Théorie des Possibilités. Masson, Paris, 1985. 423
5. D. Dubois, H. Fargier, H. Prade: Decision–Making under Ordinal Preferences and Comparative Uncertainty. In: UAI 97 – Proceedings of the 13th Annual Conference (Dan Geiger and Prakash Shenoy, Eds.). 1997, pp. 157–164. 430
6. D. Dubois, H. Prade, R. Sabbadin: Decision under Qualitative Uncertainty with Sugeno Integrals – an Axiomatic Approach. In: IFSA'97 – Proceedings, Vol. I, pp. 441–446. 430
7. D. Dubois, H. Prade, R. Sabbadin: Qualitative Decision Theory with Sugeno Integral. In: UAI 98 – Proceedings of the 14th Annual Conference (Gregory F. Cooper and Serafim Morales, Eds.), Morgan Kaufmann Publishers, Inc., San Francisco, 1998, pp. 121–128. 430
8. I. Kramosil: Boolean-Like Interpretation of Sugeno Integral. In: ECSQARU 99 – Proceedings (Anthony Hunter and Simon Parsons, Eds.). LNAI 1638, Springer Verlag 1999, pp. 245–255. 426
9. I. Kramosil: On Statistical and Possibilistic Independence. Submitted for publication. 425
10. E. L. Lehman: Testing Statistical Hypotheses. John Wiley and Sons, New York, 1947. 423
11. L. A. Zadeh: Fuzzy Sets as a Basis for a Theory of Possibility. Fuzzy Sets and Systems 1 (1978), no. 1, pp. 3–28. 423

Quantum Finite One-Counter Automata*

Maksim Kravtsev

Department of Computer Science, University of Latvia,
Raina bulv. 19, Riga, Latvia,
maksims@batsoft.lv

Abstract. In this paper the notion of quantum finite one-counter automata (QF1CA) is introduced. Introduction of the notion is similar to that of the 2-way quantum finite state automata in [1]. The well-formedness conditions for the automata are specified ensuring unitarity of evolution. A special kind of QF1CA, called simple, that satisfies the well-formedness conditions is introduced. That allows specify rules for constructing such automata more naturally and simpler than in general case. Possible models of language recognition by QF1CA are considered. The recognition of some languages by QF1CA is shown and compared with recognition by probabilistic counterparts.

1 Introduction

Quantum computation has proved to be of great interest for current and future researches. In the quantum computation theory many counterparts of the basic constructions of the classical computation theory such as quantum 1-way and 2-ways automata, quantum Turing machines etc. have been defined. It has been proved that some of these constructions are more powerful than their classical counterparts (see [2], page 150 for more details). The aim of this paper is to introduce a quantum counterpart for one-counter finite automata of the classical computation theory, namely quantum finite one-counter automata, and compare it with deterministic and probabilistic cases.

The definition of introduced automata and its well-formedness conditions are quite similar to that of 2-way quantum automata in [1] and [2]. We introduce also a special case of QF1CA called simple, with conditions, that allow construct such automata easier than in general case. Some features of language recognition by the automata are considered. Recognition of some languages with the introduced automata, namely 0^n10^n, $0^n10^n10^n$, 0^n1^n and $0^l10^m10^n$ $\{exactly\ 2\ of\ l,m,n\ are\ equal\}$, $0^l10^m10^n$ $\{(l = n\ or\ m = n)\ and\ \neg(l = m)\}$ is shown.

* Supported by Grant No. 96.0282 from the Latvian Council of Science, Grant No. 1067 from the Sponsorship Programme of Latvian Fund of Education for Education, Science, and Culture.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 431–440, 1999.

2 Classical One-Counter Automata

Definition 1. *A one-counter deterministic finite automaton (DF1CA) A is specified by the finite (input) alphabet Σ, the finite set of states Q, the initial state q_0, the sets $Q_a \subset Q$ and $Q_r \subset Q$ of accepting and rejecting states, respectively, with $Q_a \cap Q_r = \emptyset$, and the transition function $\delta\colon Q \times \Sigma \times S \to Q \times \{\leftarrow\downarrow\rightarrow\}$, where $S = \{0, 1\}$.*

There is a counter that can contain an arbitrary large integer value, it is 0 at the beginning of computation. $\leftarrow, \downarrow, \rightarrow$, respectively, decreases by one, retains the same and increases by one the value of the counter. S is defined to be 0 if and only if the value of the counter is equal to 0, otherwise $S = 1$.

The computation of DF1CA on the input word $x \in \Sigma^*$ can be described as follows. We assume that word is written on the tape. The automaton (in the state q) reads a letter of the word written on the tape (σ), checks the value of the counter (s), finds an appropriate value of the transition function $\delta(q, \sigma, s) \to q', d$. Then the state of the automaton changes to a new state q' and the value of the counter changes according to value of d as described above. Then the computation continues with the next letter of the word. After reaching of the end of the word if the automaton is in the accepting state, then the automaton accepts the word, if it is in rejecting state then the automaton rejects the word.

Definition 2. *A probabilistic finite one-counter automaton (PF1CA) A is specified by the finite (input) alphabet Σ, the finite set of states Q, the initial state q_0, the sets $Q_a \subset Q$ and $Q_r \subset Q$ of accepting and rejecting states, respectively, with $Q_a \cap Q_r = \emptyset$, and the transition function $\delta\colon Q \times \Sigma \times S \times Q \times \{\leftarrow\downarrow\rightarrow\} \to R^+$, where $S = \{0, 1\}$ and δ satisfies the following condition:*

$$\sum_{q',d} \delta(q, \sigma, s, q', d) = 1 \quad \text{for each } q, q' \in Q, \sigma \in \Gamma, s \in \{0, 1\}, d \in \{\leftarrow, \downarrow, \rightarrow\}.$$

Example 1. DF1CA can recognize, for example, the language L_1 $0^n 1 0^n$.

Example 2. It is known that PF1CA can recognize the language L_2 $0^n 1 0^n 1 0^n$ with probability $1 - 1/n$, for each $n \in N, n \geq 2$. The basic idea of the automaton is that the probabilistic decision is made during the first step and one of the following n paths is chosen with equal probability. Each path is a deterministic automaton that accepts the word if it is in the form $0^i 1 0^j 1 0^k$ and an equation in the form $a * i + b * j = (a + b) * k$, where $a, b \in N$ is satisfied. We can choose such a, b for each path that the equation can be satisfied at most in one path for any word, which is in form $0^i 1 0^j 1 0^k$ and does not belong to L_2. Thus if the word belongs to L_2 than the automaton accepts it with probability 1. If the word is not like $0^i 1 0^j 1 0^k$ than it is rejected with probability 1. If the word does not belong to L_2 but is like $0^i 1 0^j 1 0^k$ then it is rejected with probability at least $1 - 1/n$.

Now we can consider the quantum counterpart.

3 Quantum Finite One-Counter Automata

Definition 3. *A quantum finite one-counter automaton (QF1CA) A is specified by the finite (input) alphabet Σ, the finite set of states Q, the initial state q_0, the sets $Q_a \subset Q$ and $Q_r \subset Q$ of accepting and rejecting states, respectively, with $Q_a \cap Q_r = \emptyset$, and the transition function $\delta\colon Q \times \Gamma \times S \times Q \times \{\leftarrow\downarrow\rightarrow\} \to C$, where $\Gamma = \Sigma \cup \{\#, \$\}$ is the tape alphabet of A and symbols $\#, \$$ are endmarkers not in Σ, $S = \{0, 1\}$, and δ satisfies the following conditions (of well-formedness) for each $q_1, q_2, q' \in Q, \sigma \in \Gamma, s \in \{0, 1\}, d \in \{\leftarrow, \downarrow, \rightarrow\}$:*

1. *Local probability and orthogonality condition*

$$\sum_{q',d} \delta^*(q_1, \sigma, s_1, q', d)\delta(q_2, \sigma, s_2, q', d) = \begin{cases} 1, & \text{if } q_1 = q_2 \\ 0, & \text{if } q_1 \neq q_2 \end{cases}$$

2. *Separability condition I*

$$\sum_{q',d} \delta^*(q_1, \sigma, s_1, q', \rightarrow)\delta(q_2, \sigma, s_2, q', \downarrow) + \sum_{q',d} \delta^*(q_1, \sigma, s_1, q', \downarrow)\delta(q_2, \sigma, s_2, q', \leftarrow) = 0$$

3. *Separability condition II*

$$\sum_{q',d} \delta^*(q_1, \sigma, s_1, q', \rightarrow)\delta(q_2, \sigma, s_2, q', \longleftarrow) = 0,$$

where * denotes complex conjunctive.

Formally $A = (\Sigma, Q, q_0, Q_a, Q_r, \delta)$.

In order to process an input word $x \in \Sigma^*$, we assume that the word is written on the tape in the form $w_x = \#x\$$. The definition of a counter and actions with it remain the same as for deterministic automaton.

We will prove that the evolution of a QF1CA A, satisfying these conditions, is unitary.

For an integer n let C_n be the set of all possible configurations of A for inputs of length n. The definition determines that at the n-th step automata reads n-th symbol of w_x, and before the n-th step the counter can contain value from $-(n-1)$ up to $n-1$. So the configuration of A for each specific input x at each step can be uniquely determined by a pair $(q, k), q \in Q$ and $k \in [0, n-1]$, where q is the state of the automata and k is value of the counter.

A computation of A on an input x of length n corresponds to a unitary evolution in the underlying Hilbert space $H_{A,n} = l_2(C_n)$ (see [2] for more details). For each $c \in C_n, |c\rangle$ denotes the basis vector in $l_2(C_n)$, we'll use also $|q, k\rangle$. Each state in $H_{A,n}$ will therefore have a form $\sum_{c \in C_n} \alpha_c |c\rangle$, where $\sum_{c \in C_n} |\alpha_c|^2 = 1$. The automaton A induces for any input $x \in \Sigma^n$ a linear operator U_x^δ that is defined as $U_x^\delta |q,k\rangle = \sum_{q',d} \delta(q, w_{xi}, sign(k), q', d)\,|q', k+\mu(d)\rangle$ for a configuration $(q,k) \in C_n$, where w_{xi} denotes i-th symbol of $w_x = \#x\$, \mathrm{sign}(k) = 0$ if $k = 0$ and 1 otherwise, $\mu(d) = -1(0)[1]$ if $d = \leftarrow(\downarrow)[\rightarrow]$. By linearity U_x^δ is extended to map any superposition of basis states.

Theorem 1. *For any input string x the mapping U_x^δ is unitary if and only if the conditions (1) to (3) of Definition 3 are satisfied.*

Proof. To prove the Theorem, it is necessary to show that these conditions correspond to unitarity conditions of U_x^δ, that can be rewritten as:

1. $\left\| U_x^\delta |q,k\rangle \right\| = \left\langle U_x^\delta |q,k\rangle \, | U_x^\delta |q,k\rangle \right\rangle = 1$, for all configurations (q,k);
2. $U_x^\delta |q_1,k_1\rangle \perp U_x^\delta |q_2,k_2\rangle$ for all different configurations (q_1,k_1) and (q_2,k_2) or the same $\left\langle U_x^\delta |q_1,k_1\rangle \, | U_x^\delta |q_2,k_2\rangle \right\rangle = 0$ where $q, q_1, q_2 \in Q$ and $k, k_1, k_2 \in [0, |x|+1]$.

$$\left\langle U_x^\delta |q_1,k_1\rangle \, | U_x^\delta |q_2,k_2\rangle \right\rangle = \sum_{q',d_1,d_2} \delta^*(q_1, w_{xi}, sign(k_1), q', d_1)\delta(q_2, w_{xi}, sign(k_2), q', d_2)$$

where $k_1 + d_1 = k_2 + d_2$. Each member of the sum corresponds to the product of the amplitudes of $U_x^{\delta *} |q_1,k_1\rangle$ and $U_x^\delta |q_2,k_2\rangle$ mapping to the same configuration $|q', k_1+d_1\rangle = |q', k_2+d_2\rangle$.

If conditions 1. to 3. of Definition 3 are satisfied, then

1. $\left\| U_x^\delta |q,k\rangle \right\| = \sum_{q',d} \delta^*(q, w_{xi}, sign(k), q', d)\delta(q, w_{xi}, sign(k), q', d) = 1$, when 1. is true for $q_1 = q_2$
2. We observe separately the following cases $(k_1 \le k_2)$ (for $(k_2 < k_1)$ it can be shown similar):
   2.1. $k_1 = k_2$ $(q_1 \ne q_2)$
   $$\sum_{q',d} \delta^*(q_1, w_{xi}, sign(k), q', d)\delta(q_2, w_{xi}, sign(k), q', d) = 0$$
   when 1. is true for $q_1 \ne q_2$.
   2.2. $k_2 - k_1 > 2$ There is no such $|q', k'\rangle$ for which there is non zero amplitude in both $U_x^{\delta *} |q_1,k_1\rangle$ and $U_x^\delta |q_2,k_2\rangle$ in this case, because the value of the counter can change at most by 1 at each step.
   2.3. $k_2 - k_1 = 2$
   $$\sum_{q'} \delta^*(q_1, w_{xi}, sign(k_1), q', \rightarrow)\delta(q_2, w_{xi}, sign(k_2), q', \leftarrow) = 0$$
   if 3. is true.

2.4. $k_2 - k_1 = 1$

$$\sum_{q'} \delta^*(q_1, w_{xi}, sign(k_1), q', \downarrow)\delta(q_2, w_{xi}, sign(k_2), q', \leftarrow) +$$
$$\sum_{q'} \delta^*(q_1, w_{xi}, sign(k_1), q', \rightarrow)\delta(q_2, w_{xi}, sign(k_2), q', \downarrow) = 0$$

when 2. is true. □

We see that these conditions mathematically are quite similar to those for 2-way quantum finite automata from [1], but logically they express quite different processes. For the automata with counter the position of the head is determined at each step and computation takes exactly the length of input word steps. But at the same time the changing of the counter value is not deterministic.

Conditions 1–3 are not very easy to test for the concrete automaton. So like the definition of simple 2-way QFA, see [1], we can define *simple QF1CA.*

Definition 4. *A QF1CA is simple, if for each $\sigma \in \Gamma, s \in \{0, 1\}$ there is a linear unitary operator $V_{\sigma,s}$ on the inner product space $l_2(Q)$ and a function $D\colon Q, \Gamma \rightarrow \{\leftarrow, \downarrow, \rightarrow\}$ such that for each $q \in Q, \sigma \in \Gamma, s \in \{0, 1\}$*

$$\delta(q, \sigma, s, q', d) = \begin{cases} \langle q'|V_{\sigma,s}|q\rangle & \text{if } D(q', \sigma) = d \\ 0 & \text{else} \end{cases}$$

where $\langle q'|V_{\sigma,s}|q\rangle$ denotes the coefficient of $|q'\rangle$ in $V_{\sigma,s}\,|q\rangle$.

Theorem 2. *A simple QF1CA satisfies the well-formedness conditions 1–3 if and only if*

$$\sum_{q'} \langle q'|V_{\sigma,s}|q_1\rangle^* \, \langle q'|V_{\sigma,s}|q_2\rangle = \begin{cases} 1, \text{if } q_1 = q_2 \\ 0, \text{if, } q_1 \neq q_2 \end{cases}$$

for each $\sigma \in \Gamma, s \in \{0, 1\}$. That holds if and only if every operator is unitary.

Proof. We can simply rewrite well-formedness conditions:

$$\sum_{q',d} \delta^*(q_1, \sigma, s, q', d)\delta(q_2, \sigma, s, q', d) =$$
$$\sum_{q'} \delta^*(q, \sigma, s, q', D(q', d))\delta(q, \sigma, s, q', D(q', d)) + 0 =$$
$$\sum_{q'} \langle q'|V_{\sigma,s}|q_1\rangle^* \, \langle q'|V_{\sigma,s}|q_2\rangle$$

Conditions 2. and 3. are satisfied because $D(q', d)$ can be equal only to one of the ←, ↓, → for all δ in the sum. But each member of the sums 2., 3. has two multipliers with different d (←, ↓ and ↓, → in 2. and ←, → in 3.). So each member of the sum always has at least one of the multipliers equal to 0. Thus the whole sum is 0 too. We can use these considerations also to prove the theorem in the opposite direction. □

Now we can define how computation of QF1CA proceeds.

4 Language Recognition for QF1CA

Acceptance and rejection can be defined for QF1CA in some ways similarly as for push-down quantum finite automata (see [4]).

An observable used in QF1CA is defined like the description of observation of 2-way quantum automata [2], page 158. The difference is that here acceptance can be defined in 3 different ways:

1. acceptance both by state and zero value of the counter
2. acceptance by zero value of the counter
3. acceptance by state

For each input word x with $n = |x|$ and a QF1CA $A = (\Sigma, Q, q_0, Q_a, Q_r, \delta)$ let

$$C_n^a = \left\{ (q,k) \mid (q,k) \in C_n, \begin{array}{l} 1.\ q \in Q_a, k = 0 \\ 2.\ k = 0 \\ 3.\ q \in Q_a \end{array} \right\}$$

$$C_n^r = \left\{ (q,k) \mid (q,k) \in C_n, \begin{array}{l} 1.\ q \in Q_r \\ 2.\ q \in Q_r, k \neq 0 \\ 3.\ q \in Q_r \end{array} \right\}$$

and $C_n^{-} = C_n - C_n^a - C_n^r$. Let E_a, E_r and E_{-} be the subspaces of $l_2(C_n)$ spanned by C_n^a, C_n^r and C_n^{-} respectively.

The "computational observable" Ω corresponds to the orthogonal decomposition $l_2(C_n) = E_a \oplus E_r \oplus E_{-}$. The outcome of any observation will be either "accept" (E_a) or "reject" (E_r) or "non-terminating" (E_{-}).

The language recognition by A is now defined as follows: for an $x \in \Sigma^*$ as the input is used $w_x = \#x\$$, and computation starts in the state $|q_0, 0\rangle$ – counter is set to 0. For each letter from w_x operator U_x^δ is applied to current state and the resulting state is observed using the computational observable Ω defined above. After it the state collapses into E_a, or E_r or E_{-}. If "non-terminating" state is observed than computation continues with next letter. The probability of the acceptance, rejection and non-terminating at each step is equal to the square of amplitude of new state for the corresponding subspace. Computation stops either after halting state is observed or word is proceeded.

Another approach is to define that measurement will be made only after applying operators U_x^δ for each letter from w_x to initial state.

For these two different approaches of definition of 1-way QFA it has been proved (see [2] page 152) that MO (measure one) QFA can be simulated by MM (measure many) QFA, but opposite is not true. For the QF1CA it can be shown too.

Theorem 3. *MO QF1CA $A = (\Sigma, Q, q_0, Q_a, Q_r, \delta)$ automaton accepting by state or by state and zero value of counter can be simulated by the MM QF1CA $A_1 = (\Sigma, Q_1, q_0, Q_{a1}, Q_{r1}, \delta_1)$ with the same acceptance type.*

Proof. To prove the theorem, it is sufficient to show that no measurement in A_1 that is made before measurement after \$ disturbs the actual state of the automaton. No disturb means that amplitudes for E_a, and E_r are both 0.

The states of A_1 are constructed by adding the same count of states to Q. $Q_1 = Q \cup Q'$. Accepting and rejecting states for the new automaton are defined to be only from Q'. $Q'_a = \{q'_i \mid q_i \in Q_a\}$, $Q'_r = \{q'_i \mid q_i \in Q_r\}$. Transition function δ_1 is defined as $\delta_1(q_1, \sigma, s, q_2, d) = \delta(q_1, \sigma, s, q_2, d)$ and $\delta_1(q_1, \$, s, q'_2, d) = \delta(q_1, \$, s, q_2, d)$ for all $q_1, q_2 \in Q$, $\sigma \in \Gamma | \$, s \in \{0, 1\}, d \in \{\leftarrow, \downarrow, \rightarrow\}$. Remaining part of δ must be defined so that δ_1 satisfies well-formedness conditions 1–3.

It can be done by defining δ_1 for all $q'_1 \in Q'$, $\sigma \in \Gamma | \$, s \in \{0, 1\}, d \in \{\leftarrow, \downarrow, \rightarrow\}$ as $\delta_1(q'_1, \sigma, s, q'_1, d) = 1$, $\delta_1(q'_1, \$, s, q_2, d) = \delta_1(q_1, \$, s, q_2, d)$, $\delta_1(q'_1, \$, s, q'_2, d) = 0$, $\delta_1(q_1, \sigma, s, q_1, d) = 0$.

Configuration of A_1 is equal to that of A during processing of the same arbitrary input word before the final \$, due to definition of δ_1, equal to δ. No measurement disturbs the state of A_1 during it, because all $q \in Q$ are non-halting. When processing \$ the final measurement of A_1 accepts and rejects the word with the same probability as A, due to $\delta_1(q_1, \$, s, q'_2, d) = \delta(q_1, \$, s, q_2, d)$ and definition of accepting and rejecting states for A_1. □

But for the acceptance by zero counter only this construction does not work, because of halting when counter is equal to 0. The opposite statement that MM QF1CA can be simulated by MO QF1CA is false. So we consider only MM QF1CA further.

The acceptance of language L for QF1CA can be defined in the same way as for other classes of automata (see [2], page 152). A QF1CA is said to accept a language L with probability $\frac{1}{2} + \varepsilon, \varepsilon > 0$, if it accepts (halts in the accepting state after measurements) with probability at least $\frac{1}{2} + \varepsilon$ and rejects any $x \notin L$ with probability at least $\frac{1}{2} + \varepsilon$.

5 Negative Values of the Counter

It is not clear which definition of the counter is more natural –

1. counter that holds only non-negative values
2. or counter that holds arbitrary integer values

For QF1CA like for PF1CA and DF1CA it can be proved that any automaton of kind 2 can be simulated by another automaton of kind 1.

Theorem 4. *Each QF1CA* $A = (\Sigma, Q, q_0, Q_a, Q_r, \delta)$ *that allows transition of kind* $\delta_1(q_1, \sigma, 0, q_2, \leftarrow) \neq 0$ *can be simulated with another QF1CA* $A_1 = (\Sigma, Q_1, q_0, Q_{a1}, Q_{r1}, \delta_1)$ *without such transitions.*

Proof. Q_1 will be defined by adding to Q a new duplicate of states Q'. $Q_{a1} = \{q_i \text{ and } q'_i \mid q_i \in Q_a\}$, $Q'_r = \{q_i \text{ and } q'_i \mid q_i \in Q_r\}$. Transition function δ_1 is defined as: $\delta_1(q_1, \sigma, s, q_2, d) = \delta(q_1, \sigma, s, q_2, d)$ if $s \neq 0$ or $d \neq \leftarrow$;

$\delta_1(q_1', \sigma, s, q_2', d) = \delta(q_1, \sigma, s, q_2, -d)$ if $s \neq 0$ or $d \neq \rightarrow$; $\delta_1(q_1, \sigma, 0, q_2', \rightarrow) = \delta(q_1, \sigma, 0, q_2, \leftarrow)$; $\delta_1(q_1', \sigma, 0, q_2, \rightarrow) = \delta(q_1, \sigma, 0, q_2, \leftarrow)$ for all $q_1, q_2 \in Q, q_1', q_2' \in Q', \sigma \in \mathit{\Gamma}, s \in \{0,1\}, d \in \{\leftarrow, \downarrow, \rightarrow\}$. All the other values of δ_1 are defined as 0.

We can rewrite conditions 1–3 and see that they are satisfied for the new automaton. It is easy to see that computing of A_1 differs from one of A by A_1 contains configurations $|q', k\rangle$ instead of A configurations $|q', -k\rangle$ for each $k < 0$.

□

6 Languages Recognizable by QF1CA

Example 3. Simple QF1CA recognizing $L_1 = 0^n 1 0^n$.
$Q = \{q_0, q_1, q_2, q_a, q_r, q_{r2}\}, Q_a = \{q_a\}, Q_r = \{q_r\}$. q_0 is the initial state. δ is specified by transitions: $V_{\#,0}|q_0\rangle = |q_1\rangle, V_{0,s}|q_1\rangle = |q_1\rangle, V_{1,s}|q_1\rangle = |q_2\rangle, V_{0,s}|q_2\rangle = |q_2\rangle, V_{\$,s}|q_1\rangle = |q_r\rangle, V_{\$,1}|q_2\rangle = |q_r\rangle, V_{\$,0}|q_2\rangle = |q_a\rangle$ (where $V_{\sigma,s}|q\rangle$ denotes δ for $\sigma \in \mathit{\Gamma}, s \in \{0,1\}, q \in Q$) and the remaining transitions are defined arbitrary so that unitarity requirements are satisfied. $D(q_1, \#) = \downarrow$, $D(q_1, 0) = \rightarrow$, $D(q_2, 0) = \leftarrow$, $D(q_2, 1) = \downarrow$, $D(q_a, \sigma) = \downarrow$, $D(q_r, \sigma) = \downarrow$, $D(q_{r2}, \sigma) = \downarrow$.

So the automaton starts from q_0 and moves to q_1. While reading 0 it remains in q_1 and increases counter, after 1 it moves to q_2 and then while reading 0 remains in q_2 and decreases counter. If the counter is 0 when state is q_2 and 0 is read, then the word is rejected, when \$ is read denoting the end of the word if counter is 0 then the word is accepted else rejected.

Actually it is a deterministic reversible automaton. The automaton accepts the word in L_1 with probability 1 and rejects words that are not in L_1 with probability 1. The automaton is in accepting state only when counter value is 0 before reading \$. So any of the acceptance types can be used. The automaton can be easily modified to work only with nonnegative counter value (transition $V_{0,s}|q_2\rangle = |q_2\rangle$ must be substituted by $V_{0,1}V_{\#,0}|q_2\rangle = |q_2\rangle, V_{0,0}|q_2\rangle = |q_r\rangle$, non specified transitions can be still defined arbitrary so that unitarity requirements are satisfied).

Example 4. Simple QF1CA recognizing $0^n 1^n$ by state and zero value.
The construction is slightly modified construction for 1-way QFA recognizing $0^i 1^j$ given in [3] – further 1QFA($0^i 1^j$). $Q = \{q_0, q_1, q_2, q_a, q_r, q_{r2}\}$, q_0 is the initial state, $Q_a = \{q_a\}$, $Q_r = \{q_r\}$. δ is specified by transitions: $V_{\#,0}|q_0\rangle = \sqrt{1-p}\,|q_1\rangle + \sqrt{p}\,|q_2\rangle$, $V_{0,s}|q_1\rangle = (1-p)\,|q_1\rangle + \sqrt{p(1-p)}\,|q_2\rangle + \sqrt{p}\,|q_r\rangle$, $V_{0,s}|q_2\rangle = \sqrt{p(1-p)}\,|q1\rangle + p\,|q_2\rangle + \sqrt{1-p}\,|q_r\rangle$, $V_{1,s}|q_1\rangle = |q_r\rangle$, $V_{1,1}|q_2\rangle = |q_2\rangle$, $V_{1,0}|q_2\rangle = |q_{r2}\rangle$, $V_{\$,s}|q_1\rangle = |q_r\rangle$, $V_{\$,1}|q_2\rangle = |q_{r2}\rangle$, $V_{\$,0}|q_2\rangle = |q_a\rangle$ and the remaining transitions are defined arbitrary so that unitarity requirements are satisfied. $D(q_1, \#) = \downarrow$, $D(q_2, \#) = \downarrow$, $D(q_1, 1) = \leftarrow$, $D(q_2, 1) = \leftarrow$, $D(q_1, 0) = \rightarrow$, $D(q_2, 0) = \rightarrow$, $D(q_a, \sigma) = \downarrow$, $D(q_r, \sigma) = \downarrow$.

The computation of the automaton is shown in [3], there are added only counter changes, that ensure recognition of equality of the count 0 and 1. The probability p of acceptance is the same as for 1QFA($0^i 1^j$) the root of the equation $p = 1 - p^3 = 0.68\ldots$

Example 5. Simple QF1CA recognizing $L_2 = 0^n10^n10^n$ by state and zero value. Simple QF1CA can be defined similar to PF1CA of Example 2. $V_{\#,0}\,|q_0\rangle = \sum_{1..n} \frac{1}{\sqrt{n}}\,|q_i\rangle$, $D(q_i, \#) = \downarrow$. All the other transitions are strictly deterministic for each of q_1, q_n. For example further transitions for $a = 1, b = 0$ from Example 2 look like: $V_{0,s}(q_1) = q_1$, $D(q_1, 0) = \rightarrow$; $V_{1,s}(q_1) = q'_2$, $D(q'_2, 1) = \downarrow$; $V_{0,s}(q'_2) = q'_2$, $D(q'_2, 0) = \downarrow$; $V_{1,s}(q'_2) = q'_3$, $D(q'_3, 1) = \downarrow$; $V_{0,s}(q'_3) = q'_3$, $D(q'_3, 0) = \leftarrow$; $V_{\$,0}(q'_3) = q'_{acc}$, $D(q'_{acc}, 1) = \downarrow$; $V_{\$,1}(q'_3) = q'_{rej}$, $D(q'_{rej}, 1) = \downarrow$; all the other transitions for each "path" can be specified to reject the word and keep $V_{\sigma,s}$ unitary. Some additional rejecting states are necessary for it.

This automaton acts the same as corresponding PF1CA. Thus the probability of recognizing L_2 by it is the same as for PF1CA.

7 Recognizing L3 and L4

Theorem 5. *There is an QF1CA A that recognizes language $L_3 = 0^l10^m10^n$ $((l = n$ or $m = n)$ and $\neg(l = m))$ by state and zero counter with probability $\frac{4}{7}$.*

Proof. Let $V_{\#,0}\,|q_0\rangle = \sqrt{\frac{2}{7}}\,|q_1\rangle + \sqrt{\frac{2}{7}}\,|q_2\rangle + \sqrt{\frac{3}{7}}\,|q_a\rangle$ where q_0 is initial state, q_1, q_2 non-terminating states, q_a accepting state. Transitions for q_1 and q_2 for every $\sigma \in \Gamma|\$$ can be defined in such way, that each of them would be reversible but deterministic and the following conditions are satisfied:

1. If the word is not like $0^i10^j10^k$ than the word is rejected in each path.
2. If the word is of that kind that no rejection or acceptance occur during the computation. The first path leads to the state q'_1 and counter equal to the $m - n$, the second to q'_2 and $l - n$ (such a computation is easy to build like one from the Example 5).

So before reading \$ word is accepted with $p = \frac{3}{7}$ (during the first step), rejected if it is not of kind $0^i10^j10^k$ with probability $1 - \frac{3}{7} = \frac{4}{7}$, and is in the superposition $|q'\rangle = \sqrt{\frac{1}{2}}\,|q'_1, l - n\rangle + \sqrt{\frac{1}{2}}\,|q'_2, m - n\rangle$ otherwise.

If $V_{\$,0}\,|q'_1\rangle = \frac{1}{\sqrt{2}}\,|q_a\rangle - \frac{1}{\sqrt{2}}\,|q_r\rangle$; $V_{\$,0}\,|q'_2\rangle = -\frac{1}{\sqrt{2}}\,|q_a\rangle - \frac{1}{\sqrt{2}}\,|q_r\rangle$; $D(\$, q_a) = \downarrow$; $V_{\$,1}\,|q'_1\rangle = -\frac{1}{\sqrt{2}}\,|q_a\rangle + \frac{1}{\sqrt{2}}\,|q_r\rangle$; $V_{\$,1}\,|q'_2\rangle = \frac{1}{\sqrt{2}}\,|q_a\rangle - \frac{1}{\sqrt{2}}\,|q_r\rangle$ then after \$ we get:

1. If $l = m = n$ then the state is $(\frac{1}{2} - \frac{1}{2})\,|q_a, 0\rangle + (-\frac{1}{2} - \frac{1}{2})\,|q_r, 0\rangle$. The word is rejected with $p = 1$, and the total probability of rejection is $1 - \frac{3}{7} = \frac{4}{7}$.
2. If all l, m, n are different, then the word is rejected with probability $p = 1$, because the counter value is not 0 for both $|q'_1, l - n\rangle$ and $|q'_2, m - n\rangle$.
3. $(l = n)$ and $\neg(l = m)$ then the state is $\frac{1}{2}\,|q_a, 0\rangle - \frac{1}{2}\,|q_r, 0\rangle - \frac{1}{2}\,|q_a, m - n\rangle - \frac{1}{2}\,|q_r, m - n\rangle$. So the word is accepted at the \$ with $p = \frac{1}{4}$ and total probability of accepting of the word $p = \frac{1}{4} * \frac{4}{7} + \frac{3}{7} = \frac{4}{7}$.
4. $(m = n)$ and $\neg(l = m)$. The same as shown in the previous item.

So the automaton recognizes L_3 with probability $\frac{4}{7}$. □

Theorem 6. *There is an QF1CA A that recognizes language* $L_4 = 0^l 10^m 10^n$ *(exectly two of* l, m, n *are equal) with probability* $\frac{1}{2}+\epsilon$.

Proof. Omitted. Similar to one for the Theorem 5.

References

1. Kondacs, A., Watrous, J.: On the power of quantum finite state automata. In Proc. 38th FOCS (1997), 66–75. 431, 431, 435, 435
2. Gruska, J.: Quantum Computing, McGraw Hill (1999). 431, 431, 433, 436, 436, 437
3. Ambainis, A., Freivalds, R.: 1-way quantum finite automata: strengths, weaknesses and generalizations. In Proc. 39th FOCS (1998), 332–341. 438, 438
4. Moore, C., Crutchfield, J. P.: Quantum Automata and Quantum Grammars http://xxx.lanl.gov/quant-ph/970731. 436

A Performance Comparison of Mobile Agents and RPC[*]

David Rutter

Department of Computing Science
Chalmers University of Technology
Göteborg, Sweden
rutter@cs.chalmers.se

Abstract. The issue of Mobile Agent performance, and in particular, scalability is one that, while receiving attention in a speculative sense, has not been addressed to any great extent in empirical studies. This paper provides an insight into the requirements that an MA system has on a server's resources, comparing them to the requirements for an equivalent RPC solution. The basis of the study is an experiment that measures timing and system load information (CPU, memory, network and disk IO) as an MA and RPC solution performs a simple task.

1 Introduction

Mobile Agent (MA) technology is designed to combine the agency and autonomy of static agents, described in studies such as [4], with mobility, this being the ability for an agent to move to another host. It is these concepts of agency and autonomy that distinguishes MAs from technologies such as mobile code and remote evaluation.

One of the primary motivations for the development of MA systems has been the arguments put forward by authors such as Chess, Harrison and Kershenbaum [7], and White [12] in favour of using MAs over other distributed processing technologies such as Remote Procedure Call (RPC). These papers [7,12] propose a number of performance benefits that can be gained through the use of MAs. These claims were largely speculative, and were made at a time when few MA systems were available. Even so, given the number of MA systems developed since these studies, there have been relatively few empirical studies that assess MA performance in light of these claims.
In short these claimed performance benefits are:

- reduced network load and execution time, as they are able to migrate to the site of the data.
- increased scalability over technologies such as Remote Procedure Call (RPC), as MAs are able to operate asynchronously.

[*] This research was conducted while the author was studying at the Distributed Computing Laboratory, School of Computing and Information Technology, University of Western Sydney, Nepean, Australia.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 441–448, 1999.

While it is generally accepted that by moving the processing to the site of the data the network load and execution time would be reduced [2], assuming of course that the processing requirements are largely data-bound, this is a benefit that is available in other technologies, and is not solely the domain of MAs [3].

The argument in favour of increased scalability of MAs is also rather limited, as it concentrates on the assumed method of communication (asynchronous vs synchronous), and does not take into account the many other factors that influence a solution's scalability, such as the load on the server's CPU, memory and IO resources.

The definition of the term 'performance' that is used for this study follows this line of reasoning, whereby in addition to the measures of execution time and network traffic that are used in most studies comparing MAs and RPC, the issue of scalabilty will also be considered from the point of view of the relative impact that an MA or RPC has on the CPU, memory, and IO resources of the server.

The basis of the study is an experiment that captures timing and system load information (CPU, network, memory and disk IO) as each system performs a simple task. From this information we are able to gain an insight into the performance characteristics of each technology, and apply this information to cast doubt on claims of performance advantages for MAs, and to give an, admittedly rough, quantitative indication of the cost in performance terms of the other advantages claimed for mobile code.

2 Prior Studies

The research into the performance of MAs has progressed from the speculative studies conducted by Chess et al. [7], to the more detailed experiments carried out by Gray [5] and Johansen [8]. Two of the seminal papers describing mobile agent performance are by Chess et al. [7] and by White [12]. The paper by Chess et al. [7] was produced at a time when there were few working implementations of MAs and thus was largely based on what MAs had, in 1995, the potential to deliver. The paper compares MAs and RPC in a number of areas, one of which is performance. The issue of scalability is raised by the authors, in which they conjecture that since MAs operate asynchronously, and RPCs operate synchronously[1], then it is likely that an MA system would be able to support a higher transaction rate than an equivalent RPC solution.

Studies such as those by Straßer and Schwehm [11], and Chia and Kannapan [1] have focused on developing models that would allow the MA system to decide on whether it is more economical to migrate to the site of the data, or to use a naive form of RPC, and bring the data to the MAs current location. Both studies show that in most cases the selective migration approach performs better than a migrate-always, or never-migrate policy.

[1] The most basic form of RPC is synchronous. There are many implementations of RPC, some of which are asychronous. For the purpose of the study all comparisons between RPC and other technologies will be based on the synchronous form of RPC.

In most empirical studies on MA performance, MAs are compared to RPC, this can be seen in studies such as those by Johansen [8] and Straßer and Schwehm [11]. In studies such as these the comparison is usually based on the RPC solution following the naive RPC model whereby the data required by the application must be shipped in its entirety across the network – no processing is available at the server. Johansen [8] in particular uses this representation of RPC to show that MAs perform better as the size of the data increases, and as the number of clients increase. One has only to look at the manner in which clients accessing a database server submit SQL queries to the server to see that MAs are not essential to move the processing to the site of the data.

The work by Gray [5] is based on the Agent Tcl MA system. The analysis performed by Gray [5] is perhaps the most detailed available now. Gray [5] again compares MAs and RPC, as well as TCP/IP (using TCP sockets). The results gained by Gray [5] were not always encouraging, however he notes that overall there is sufficient evidence to suggest that further improvements to Agent Tcl would be worthwhile. Agent Tcl suffers from the same problem that most interpreted languages suffer, this being the slowness in performance. Gray [5] notes that the effectiveness of the MA solution is directly related to the level of CPU use.

While few studies have attempted to compare the scalability of MAs and RPC to any great degree, studies such as those by Kotz and Gray [9], and Peine and Stolpmann [10] have indicated that the scalability of MAs may not be as favourable as Chess et al. [7] had conjectured.

3 Experiment Design

A comparison is made between Java based MAs (specifically IBM Aglets) and native RPC. This might be objected to as "unfair" to MAs on grounds of the overhead imposed by the Java platform itself. However the following observations should be noted:

- the magnitude of this overhead is well known (a factor of 10–30), and so may be allowed for; particularly when, as turns out to be the case, the performance results of the two are separated by several orders of magnitude; and
- in choosing between alternative technologies to solve a problem it makes sense to use the most efficient available implementation of each. The argument may well shift to the relative systems administration overhead of installing native RPCs as opposed to that for an MA server, or even Java RMIs, but our immediate concern in this study is to investigate the extent of the price we have to pay performance-wise for any such advantage as well as providing evidence to dispute the claim that MAs actually offer a performance advantage.

3.1 Experiment Description

The basis of the experiment was to measure the impact that a single RPC or MA had on the system at different levels of server load, simulated by creating unrelated background processes.

The results were recorded for each system as they performed the task of searching through a file at the remote host, counting the number of occurrences of a given word. The task, although trivial, was designed to represent a task that would demonstrate the characteristics of a 'real world' application, i.e. a reasonable use of CPU, IO and memory resources. In contrast to many of the previous experiments comparing MAs and RPC [8], the RPC solution implemented for this experiment was not a naive RPC that could only return the data in its entirety back to the client, instead it was assumed that a more 'intelligent' RPC solution were available, and thus the RPC was capable of performing the processing on the server.

For the study a specific implementation of MAs and RPC was chosen. For the MA implementation, the Aglet Software Development Kit 1.02 (ASDK) from IBM was used. The ASDK is a Java based MA system, the entire system runs within the Java Virtual Machine (JVM). For the RPC solution the widely used Sun ONC RPC was used. The RPC was written in the C language.

While it was possible to implement the RPC solution using Java's RMI, the aim of the study was to provide a comparison based on a typical implementation of the technology. Generally an RPC solution would be developed in a language such as C or C++.

3.2 Experiment Variables

The measurements were recorded at different levels of server CPU load, these being: 0, 20, 40, 60 and 80%.
At each level of server load the following variables were recorded:

– Server load: This can be broken down into:
  - CPU
  - Memory
  - Disk IO
  - Network load
– Processing Time: This is the time required for the system to service a job (MA or RPC). This in turn can be broken down into:
  - Total response time
  - Client marshalling time
  - Server un-marshalling time
  - Server processing time
  - Server marshalling time

3.3 The Experiment Environment

The client system for the experiment was a Sun Ultra1/170, running Solaris2.6 (SunOS 5.6), with 128MB memory, and a 10-100Mbps Ethernet card. The server was a Digital Personal Workstation 500au (Miata/DECST30), running Digital UNIX 4.0D (firmware revision 6.6-29) with 256 Mb physical memory, 768 Mb swap, and a 100 Mbps full-duplex Fast Ethernet card. The network was running at 10Mbps, and both client and server were on the same network segment.

4 Results

4.1 Timing Results

The results, shown in Figures 1 and 2 below, indicate quite clearly that the fastest solution is RPC. The difference between the two solutions is large – especially so as the load on the server increases. The time for the Aglet solution ranges from 1.848 seconds for a server under no load, to 112.087 for a server under a load of 80%. In contrast the RPC time ranges from 0.02 to 0.12 seconds.

The most costly operation in the running of either solution, apart from the application processing, was the marshalling of the return result. In the case of Aglets this involves the serialisation of the Aglet and return result, in the case of the RPC solution this involves marshalling the result for transmission to the client. Both the marshalling and application processing time (the time performing the actual task) were most affected by the increase in server load.

The results for the Aglet solution were complicated by the extra operations required for the first Aglet. Once the first Aglet was received, a number of extra classes were required by the receiving server. These classes were fetched from the sending server. The impact of this was an increase in the total response time and client marshalling time, and to increase the load on the network. The impact of the first Aglet is not shown in Figure 1.

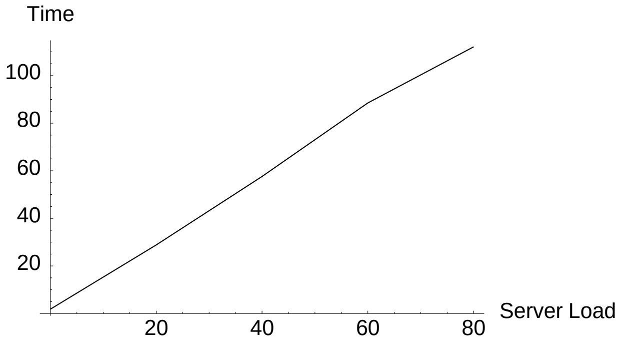


Fig. 1. Mean times (secs) for Aglet Response Time without first interaction

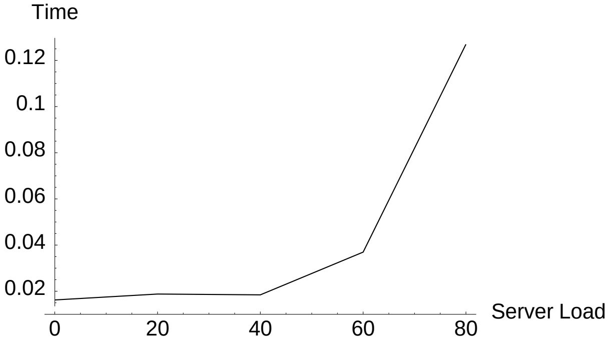


Fig. 2. Mean times (secs) for RPC Response Time

4.1.1 Latency Results. For the purpose of this study the term latency is used to describe the time that the system spends performing functions necessary for the execution of the process (Aglet or RPC), that are not the actual processing, i.e. searching through the file. This allows a measurement of the additional overhead that a particular system introduces.

As expected, for both solutions, the percentage latency decreases rapidly as the load on the server increases. This is most marked in the Aglet solution, the latency falling from 46.36% to 3.86% between the 0% and 80% load levels. Overall, the RPC solution has a much lower latency than Aglets, the percentage for RPC ranging from 1.699% to 0.44%.

4.2 Memory Use

Since the ASDK runs within the JVM, the total size of the JVM was measured. The results show that as the Aglet solution executes the size of the JVM increases from 6096K to 13000K. The memory usage within the JVM was also measured, showing that the period of greatest memory usage is during the un-marshalling and marshalling operations.

In contrast to this the memory requirements of the RPC solution were static, with a total size of 2152K.

4.3 Network Load

The Aglet solution, following the action of the first Aglet, required a mean of 92.25 input packets per Aglet. For the first Aglet, the mean number of input packets was 516.4. In contrast the RPC solution requires a mean of only 16.42 packets. This difference is due to the fact that the Aglet solution must send the Aglet and parameters (in this case the word to be searched for) to the host, whereas the RPC solution need only send the parameters necessary to execute the remote procedure.

The output results reflect a similar situation to that of the input results, with the Aglet solution requiring a mean of 62 packets, and the RPC solution requiring only 8.27 packets.

4.4 CPU Load

The differences between the CPU load for the Aglet and RPC solution become less obvious as the server load increases. At zero server load, the server load for the Aglet solution ranged from 0–0.2 and 1.4%, while for the RPC solution the range was from 0–0.1 and 0.6%. As the server load increases it does not appear that the CPU requirements of either solution increases, however for the Aglet solution the CPU is required for a longer period of time, thus producing a more sustained load on the CPU resource than the equivalent RPC solution.

4.5 IO Load

The results for the IO resource did not show any conclusive results. The RPC solution required 188 blocks of IO, whereas the Aglet solution required between 200 and 400 blocks. It should be noted that the same file was being used for both the MA and RPC.

5 Conclusion

The results show quite clearly that an RPC solution is by far the fastest in terms of raw response time. The effect of increasing the load on the server is most noticeable in the response time of the Aglet solution.

The differences in execution time can be contributed to two factors. One is the well-documented performance problems of the Java environment [6]. The second factor can only be put down to the infrastructure necessary to support the operations required to dispatch, receive, create, instantiate and execute an Aglet. Since these operations must be executed in the Java environment, the two factors are combined, and the effect becomes more and more noticeable as the load is increased.

The memory requirements of both solutions is not negligible, more so for the Aglet solution. Even though more than one Aglet can operate in the JVM, the memory required to support the Aglet system is much higher than the RPC solution, and is likely to become even more so as more Aglets are executed. The CPU requirements of the Aglet solution suggest that an Aglet solution requires more CPU resources than the RPC solution, however as with all of the server load results, the most important consideration is the time that these resources are required for the Aglet solution to perform the task.

Although the definition of ‘scalability’ is problematical in the context of the debate on the utility of MAs, the argument by Chess et al [7] that an MA system could provide a more scalable solution is not supported by the results of this experiment. There is no indication that the network resources required by the Aglet solution was significantly lower. There is also evidence to support the argument that a server is more likely to be able to support a greater number of RPC transactions – this can be seen in the CPU requirements for the Aglet solution.

References

1. Chia, T., Kannapan, S.: Strategically Mobile Agents. In: Rothermel, K., Popescu-Zeletin, R. (Eds.): Mobile Agents, Proceedings of the 1st Int. Workshop (MA '97). Lecture Notes in Computer Science, Vol. 1219. Springer-Verlag, Berlin Heidelberg New York (1997) 149–161. 442
2. Chow, R., Johnson, T.: Distributed Operating Systems and Algorithms. Addison Wesley Longman, USA (1997). 442
3. Cook, M. W., Linn, C. N., Bryan, G. M.: Mobile Agents: Essential technology or just an appealing idea? In: Proceedings of Parallel and Distributed Computing and Networks, 1998, IASTED Int. Conference. ACTA Press, Calgary, Canada (1998). 442
4. Franklin, S., Graesser, A.: Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. In: Müller, J. P., Wooldridge, M. J., Jennings, N. R. (Eds.): Intelligent Agents III: Proceedings of the 3rd Int. Workshop on Agent Theories, Architectures and Languages. Lectures Notes in Artificial Intelligence, Vol. 1193. Springer-Verlag, Berlin Heidelberg New York (1996) 21–35. 441
5. Gray, R.S.: Agent TCL : A flexible and secure mobile-agent system. Ph.D. Thesis. Dartmouth College, New Hampshire (1997). 442, 443, 443, 443, 443, 443
6. Halfhill, T.R.: How to Soup Up Java, Part 1. In: Byte, Vol 23. (1998) 60–74. 447
7. Chess, D., Harrison, C., Kershenbaum, A.: Mobile Agents: Are They a Good Idea? IBM T.J. Watson Research Centre, NY.
   Available at: http://www.research.ibm.com/massdist/mobag.ps (1995). 441, 441, 442, 442, 442, 443, 447
8. Johansen, D.: Mobile Agent Applicability. In: Rothermel, K., Hohl, F. (Eds.): Proceedings of the 2nd Int. Workshop on Mobile Agents (MA'98). Lecture Notes in Computer Science, Vol. 1477. Springer-Verlag, Berlin Heidelberg New York (1998) 80–98. 442, 443, 443, 444
9. Kotz, D., Gray, R. S.: Mobile Code: The Future of the Internet. MAC3 Workshop. Available at: http://mobility.lboro.ac.uk/MAC3 (1999). 443
10. Peine, H., Stolpmann, T.: The Architecture of the Ara Platform for Mobile Agents. In: Rothermel, K., Popescu-Zeletin, R. (Eds.): Mobile Agents, Proceedings of the 1st Int. Workshop (MA'97). Lecture Notes in Computer Science, Vol. 1219. Springer-Verlag, Berlin Heidelberg New York (1997) 50–61. 443
11. Straßer, M., Schwehm, M.: A Performance Model for Mobile Agent Systems. In: Arabnia, H. (Ed.): Proceedings of the Int. Conference on Parallel Distributed Processing Techniques and Applications (PDPTA '97), Vol II. CSREA Press. (1997) 1132–1140. 442, 443
12. White, J.: Mobile Agent White Paper. General Magic Inc. Available at: http://www.genmagic.com/technology/techwhite-paper.html (1996). 441, 441, 442

Cyclic Cutwidth of the Mesh*

Heiko Schröder[1], Ondrej Sýkora[2], and Imrich Vrťo[3]

[1] School of Applied Science, Nanyang Technological University
Nanyang Avenue, Singapore 639798, Republic of Singapore
asheiko@ntu.edu.sg

[2] Department of Computer Science, Loughborough University
Loughborough, Leicestershire, LE11 3TU, The United Kingdom
O.Sykora@lboro.ac.uk

[3] Department of Informatics, Institute for Mathematics
Slovak Academy of Sciences
P. O. Box 56, 840 00 Bratislava, Slovak Republic
vrto@savba.sk

Abstract. The cutwidth problem is to find a linear layout of a network so that the maximal number of cuts (cw) of a line separating consecutive vertices is minimized. A related and more natural problem is the cyclic cutwidth (ccw) when a circular layout is considered. The main question is to compare both measures cw and ccw for specific networks, whether adding an edge to a path and forming a cycle reduces the cutwidth essentially. We prove exact values for the cyclic cutwidths of the 2-dimensional ordinary and cylindrical meshes $P_m \times P_n$ and $P_m \times C_n$, respectively. Especially, if $m \geq n+3$, then $ccw(P_m \times P_n) = cw(P_m \times P_n) = n+1$ and if n is even then $ccw(P_n \times P_n) = n-1$ and $cw(P_n \times P_n) = n+1$ and if $m \geq 2, n \geq 3$, then $ccw(P_m \times C_n) = \min\{m+1, n+2\}$.

1 Introduction

The underlying practical problem for this paper is the tradeoff between cost and speed of computer architectures: the linear array is the least expensive architecture and the ring architecture is probably only slightly more expensive. This is dependent on the technology – in free space optics the additional connection might cost more than in fibre optics. Whether this small additional cost leads to significantly improved communication speed in the architecture depends on the communication pattern used by the parallel algorithm run on the linear array or ring (see also [3,14,16]). The communication patterns we are investigating in this paper are 2-dimensional ordinary and cylindrical meshes. There are further motivations for studying the problems: rearrangeability problems [10], VLSI design [13], isoperimetric problems [4].

* This research was supported by grant No. 95/5305/277 of Slovak Grant Agency and by grant of British Council to the Project Loughborough Reconfigurable Array and Theoretical Aspects of Interconnection Networks. The research of the last two authors was supported also by the grant of EU INCO-COP 96-0195.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 449–458, 1999.

The cutwidth problem is to find a linear layout of an interconnection network so that the number of cuts of a line separating consecutive vertices is minimized. A related and more natural problem is the cyclic cutwidth when a circular layout is considered. Both problems are NP-hard [8,12]. One of the main question is to compare both measures, denoted by *cw* and *ccw*, respectively, for specific networks, whether adding an edge to a path and forming a cycle reduces the cutwidth essentially. In [6,12], it is shown that the cyclic cutwidth equals the cutwidth in case of trees. For the toroidal mesh, the cyclic cutwidth is roughly half of the cutwidth [15]. A partial result for the cyclic cutwidth of the hypercube is in [2] showing that the cyclic cutwidth is less than 5/8 of the cutwidth. Note that another related problem is to compare the average cyclic cutwidth with the average cutwidth [5] and the cyclic bandwidth with the bandwidth [11].

In this paper we show the following results:

Theorem 1. *For $m \geq n \geq 3$*

$$ccw(P_m \times P_n) = \begin{cases} n-1, & \textit{if } m = n \textit{ is even}, \\ n, & \textit{if } n \textit{ is odd or } m = n+2 \textit{ is even}, \\ n+1, & \textit{otherwise} \end{cases}$$

and for $m \geq 2, n \geq 3$

$$ccw(P_m \times C_n) = \min\{m+1, n+2\}.$$

Actually we prove the lower bounds only. The upper bounds are in [15] as well as the remaining cases $ccw(P_m \times P_2) = 2$, for $m = 3, 4$ and $ccw(P_m \times P_2) = 3$, for $m \geq 5$. Compare the above results with the results for the cutwidth of 2-dimensional ordinary mesh: $cw(P_m \times P_n) = n + 1$, for $m \geq n \geq 3$ and $cw(P_m \times P_2) = 2$, for $m \geq 2$, and for the cutwidth of 2-dimensional cylindrical mesh: for $m \geq 2, n \geq 3$ $cw(P_m \times C_n) = 4$, if $m = 2, n = 3$ and $cw(P_m \times C_n) = \min\{2m + 1, n + 2\}$, otherwise [15].

Our results completely solve the cyclic cutwidth problem for 2-dimensional meshes. Generally saying the ratio of cw/ccw for ordinary mesh and toroidal mesh is roughly 1 and 2 respectively, while for the cylindrical mesh it lies inbetween depending on the relative sizes of m and n.

Developed method could be used to other mesh-like or product graphs.

In terms of the underlying practical problem, which hardware architecture to choose, our results are: If the communication pattern is a hypercube, torus, or "high" cylinder the ring architecture leads to a significant improvement over the linear array; if the communication pattern is a mesh or a "flat" cylinder the improvement is not significant or zero.

2 Definitions and Notations

The cyclic cutwidth is a special case of the so called congestion, an important concept from the theory of interconnection networks. Therefore it is more convenient to define it by means of the congestion. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs such that $|V_1| = |V_2|$. An embedding of G_1 in G_2 is a couple of mappings (ϕ, ψ) satisfying

$$\phi\colon V_1 \to V_2 \quad \text{is a bijection,} \ \ \psi\colon E_1 \to \{\text{set of all paths in } G_2\},$$

such that if $uv \in E_1$ then $\psi(uv)$ is a path between $\phi(u)$ and $\phi(v)$. Define the congestion of an edge $e \in E_2$ under the embedding (ϕ, ψ) of G_1 in G_2 as

$$cg(G_1, G_2, \phi, \psi, e) = |\{f \in E_1 \ : \ e \in \psi(f)\}|,$$

the congestion of G_1 in G_2 under (ϕ, ψ) as

$$cg(G_1, G_2, \phi, \psi) = \max_{e \in E_2}\{cg(G_1, G_2, \phi, \psi, e)\}$$

and the congestion of G_1 in G_2 as

$$cg(G_1, G_2) = \min_{(\phi,\psi)} \{cg(G_1, G_2, \phi, \psi)\}.$$

Let P_n denote the $n-$vertex path. Let $V_{P_n} = \{1, 2, \ldots, n\}$, with edges between i and $i+1$, for $i = 1, 2, \ldots, n-1$. Let C_n denote the $n-$vertex cycle. Vertices and edges of C_n are defined similarly.

Let $G = (V, E)$ be an $n-$vertex graph. Define the cutwidth of G as $cw(G) = cg(G, P_n)$, and the cyclic cutwidth of G as $ccw(G) = cg(G, C_n)$.

Let $P_m \times P_n$ and $P_m \times C_n$ denote the 2-dimensional ordinary and cylindrical meshes defined as the Carthesian products of two paths and a path and a cycle, respectively.

3 Lower Bounds

Our method is based on the argument proved in the Lemma 1 saying that any two pair cardinality sets of elements embedded in a ring can be cut by removing two edges of ring into two equally sized parts (similar results were also shown in [1,9]). Suppose that a mesh is embedded in a ring. As the above mentioned sets we take some specific vertices of a mesh and we construct edge disjoint paths in the mesh between the both parts of each set.

As all the paths should be embedded so that they use the two edges of ring which create the cut, the cyclic cutwidth should be at least so big as the half of number of the paths (this part of the method is well known and utilized e.g. [12,15]). In order to get exact results we improve the method according to the type of mesh.

Now we prove the lemma:

Lemma 1. *For $k, l \geq 1$, consider any $2(k+l)$ vertex cycle with vertices labelled consecutively by $0, 1, 2, \ldots, 2(k+l)-1$. Colour arbitrary $2k$ vertices by black and the rest of vertices by white. Then there exist $k+l$ consecutive vertices on the cycle containing exactly k black and l white vertices.*

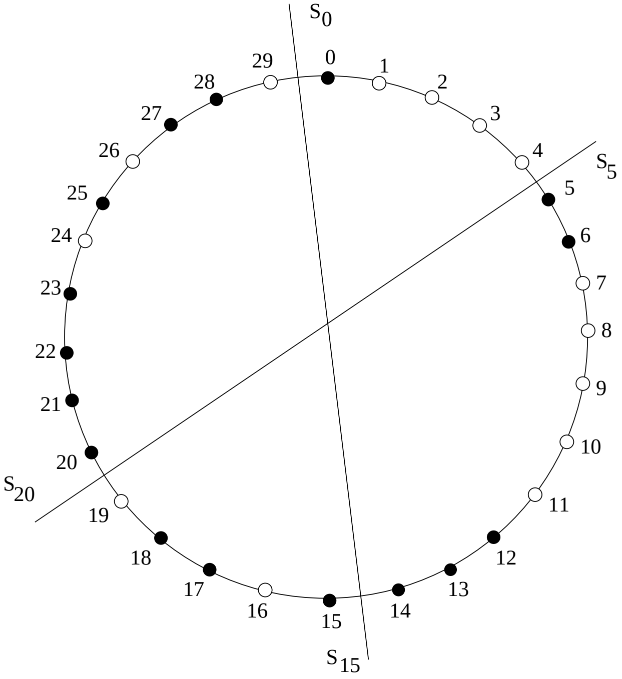


Fig. 1. Illustration of the Lemma 1 for $k = 8, l = 7$

Proof. Denote $S_i = \{i, i+1, \ldots, i+k+l-1\}$, where the arithmetic operations are taken mod $2(k+l)$. Wlog assume that S_0 contains less than k black vertices. Then S_{k+l} contains more than k black vertices. Observe that numbers of black vertices in S_i and S_{i+1} differ by at most 1. Hence there must exist an i_0, $0 < i_0 < k+l$, for which the number of black vertices in S_{i_0} equals k. See Figure 1 which illustrates the argument for $k = 8, l = 7, i_0 = 5$. □

The lower bounds in Theorem 1 will follow from the following propositions and corollaries.

Proposition 1. *For any* $n \geq 3$

$$ccw(P_n \times P_n) \geq \begin{cases} n-1, & \text{if } n \text{ is even,} \\ n, & \text{if } n \text{ is odd} \end{cases}$$

Proof. Let us have any embedding (ϕ, ψ) of $P_n \times P_n$ into the cycle C_{n^2}. Consider now only the boundary vertices of the mesh, i.e. the vertices of degree not greater than 3. Colour the embedded vertices $(i, 1), i = 1, 2, \ldots, n$ and the vertices $(n, j), j = 2, 3, \ldots, n-1$ by black and the remaining boundary vertices by white. According to lemma 1 we can cut the cycle on two edges such that each of the two paths contains exactly $n-1$ black and $n-1$ white vertices. This cut induces a cut of $P_n \times P_n$ into two parts I and II such that each part contains exactly $n-1$ black and $n-1$ white vertices. We estimate the size of the cut. Clearly the size of the cut is at least the maximum number of edge disjoint paths between I and II in the mesh. □

Consider the subgraph T_n of $P_n \times P_n$ induced by vertices (i,j), where $1 \leq j \leq i \leq n$, except for the vertex (n,n). Note that T_n contains all black vertices and no white vertex.

Claim. There are $n-1$ edge disjoint paths between the black vertices with one vertex in I and the second in II. Moreover these paths can be routed in T_n only.

Proof (of the Claim). We prove the claim by induction on n. The cases $n=3,4$ are trivial. Assume $n \geq 5$ and let the claim hold for every subgraph $T_m, 3 \leq m < n$.

If there exist two vertices $(i,1)$ and $(n,i), 1 < i < n$ belonging to different sets I and II then join the vertices by the shortest path through (i,i). Deleting the edges of the ith row and the edges of the ith column, if $i < n$, we reduce the problem to the subgraph T_{n-1}.

In the opposite case we must find i and $j, 1 < i < j < n$, such that $(i,1)$ and (n,i) belong to say I, and $(j,1)$ and (n,j) belong to II. Then join $(i,1)$ and $(j,1)$ by the path $(i,1)-(i,i)-(j,i)-(j,1)$. Similarly join (n,j) and (n,i). Deleting the edges of the ith and the jth row and column we reduce the problem to T_{n-2}. See the illustration of creating the disjoint paths on the Figure 2 for $n=8$ and a random distribution of the black vertices over I and II. The paths are shown by heavy lines. By the above procedure we get $n-3$ edge disjoint paths and T_3 in which either 2 or 3 other edge disjoint paths can be constructed. There may be constructed still other edge disjoint paths. It depends on in how many subpaths is cut the path of black vertices $(1,1)(2,1)(3,1)-(n,1)(n,2)-(n,n-1)$. If it is cut to two subpaths, we can construct altogether $n-1$ edge disjoint paths, if to three or four subpaths we can construct at least n, if to five then we can construct at least $n+1$ and if it is cut to at least six subpaths then we can construct at least $n+2$ edge disjoint paths.

Doing the same construction of edge disjoint paths with white endvertices in the subgraph induced by $(i,j), 1 \leq i \leq j \leq n$, except for the vertex $(1,1)$, we get other $n-1$ edge disjoint paths between I and II. So we have in total at least $2n-2$ edge disjoint paths between I and II in $P_n \times P_n$. This implies that the cut of $P_n \times P_n$ into the parts I and II has at least $2n-2$ edges which in turn immediately implies that the congestion is at least $(2n-2)/2 = n-1$.

If n is even we are done. In the odd case we have to strengthen the above argument. If the number of edge disjoint paths is at least $2n-1$ then we are done again. If this number is exactly $2n-2$ then the vertices $(1,j), 1 \leq j \leq n$ and $(i,1), 2 \leq i \leq n-1$ must belong to one part, say I and the remaining boundary vertices to II.

Draw the mesh in the plane in the standard way. Let G be its geometric dual graph, i.e. every face in the drawing of the mesh is replaced by a vertex and if 2 faces share an edge then the corresponding vertices are joined by an

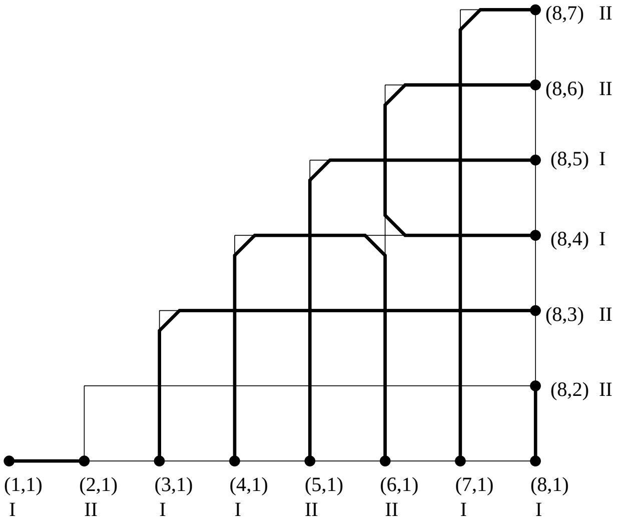


Fig. 2. Example of the edge disjoint paths in T_8

edge. Note that G is a multigraph. Let v be the vertex of G corresponding to the outer face. Let us introduce a notation $[i, j], i, j, = 1, 2, \ldots, n-1$ for the vertex of G, if it corresponds to the face $(i, j)(i+1, j)(i+1, j+1)(i, j+1)$. Note that the $(2n-2)$-edge cut of the mesh corresponds to a cycle C of length $2n-2$ in G. The cycle C contains the edge between v and $[n-1, 1]$ and between v and $[1, n-1]$. Now we repeat the whole method by changing the choice of the black vertices. Let the set of black vertices of $P_n \times P_n$ be: $(n, j), j = 1, 2, \ldots, n$ and $(i, n), i = 2, 3, \ldots, n-1$. Assume that the corresponding edge cut of $P_n \times P_n$ contains exactly $2n-2$ edges. This cut corresponds to a cycle C' in G of length $2n-2$. The cycle C' contains the edge between v and $[n-1, n-1]$ and between v and $[1, 1]$. The cycles C and C' must intersect in G in at least one vertex (k, l). Wlog we may assume that $l \geq (n+1)/2$.

Consider two shortest paths in $C \cup C'$ which contain vertices $v, [1, 1], [k, l]$ and vertices $v, [n-1, 1], [k, l]$, respectively. As $l \geq (n+1)/2$, the sum of lengths of the paths is at least $2n-1$. The edges in the paths correspond to an edge cut that separates vertices $(i, 1), i = 1, 2, \ldots, n-1$ from the rest of boundary vertices in $P_n \times P_n$. Note that some edges may be count twice. Finally, this cut corresponds to a cut on two edges of C_{n^2} s.t. one part of the cycle contains vertices $(i, 1), i = 1, 2, \ldots, n-1$ and the rest of boundary vertices is in the second part. Hence $ccw(P_n \times P_n) \geq \lceil (2n-1)/2 \rceil = n$. Illustration of the argument for $n = 7, k = 4, l = 5$ is in the Figure 3. The dual graph G is shown by heavier lines and vertices as the original mesh. Cycles C and C' are shown by the heaviest lines. □

Proposition 2. *For odd $n \geq 3$*

$$ccw(P_{n+2} \times P_n) \geq n + 1.$$

Proof. The proof is the same as above but we start with the black vertices defined to be $(i, 1), i = 2, 3, \ldots, n + 2$ and $(n + 2, j), j = 2, 3, \ldots, n$. □

The above propositions immediately imply:

Corollary 1. *For $n \geq 3$*

$$ccw(P_{n+1} \times P_n) \geq n.$$

Proposition 3. *For even $n \geq 4$*

$$ccw(P_{n+3} \times P_n) \geq n + 1.$$

Proof. Follow the proof of Proposition 1. Colour the vertices: $(i, 1), i = 3, 4, \ldots, n+3$ and $(n+3, j), j = 2, 3, \ldots, n$ by black and vertices $(1, i), i = 1, 2, \ldots, n$ and $(j, n), j = 2, 3, \ldots, n$ by white. By similar way as above we construct at least $2n$ edge disjoint paths between I and II. If there are only $2n$ such paths, i.e. exactly n edge disjoint paths in each T_{n+1}, then the vertices $(i, 1), i = 3, 4, \ldots, n+3$ and $(1, j), j = 1, 2, \ldots, n$ are in one part e.g. I and the rest of coloured vertices is in II. In this case one can easily construct a path between the vertices $(3, 1) \in I$ and $(n, n) \in II$ which is edge disjoint with the previously constructed paths. □

Now in what follows we show the lower bounds of the Theorem 1 for the cylindrical mesh.

Proposition 4. *For $n - 3 \geq m \geq 2$*

$$ccw(P_m \times C_n) \geq m + 1.$$

Proof. Taking into account the fact that $P_m \times C_n \supset P_m \times P_n$, it holds $ccw(P_m \times C_n) \geq ccw(P_m \times P_n) \geq m + 1$. □

As $P_{n+i} \times C_n \supseteq P_{n+1} \times C_n$ for any $i \geq 1$, implies $ccw(P_{n+i} \times C_n) \geq ccw(P_{n+1} \times C_n) \geq n + 2$, we need to show the following cases:
$ccw(P_m \times C_n) \geq m + 1$, where $n - 2 \leq m \leq n + 1$.

We can do it following the proof of the Proposition 1 although each case needs a special handling. For each case we define different four sets of vertices A, B, C, D and colour the vertices of A and B by black and C and D by white.

Proposition 5. *For $n \geq 4$*

$$ccw(P_{n-2} \times C_n) \geq n - 1.$$

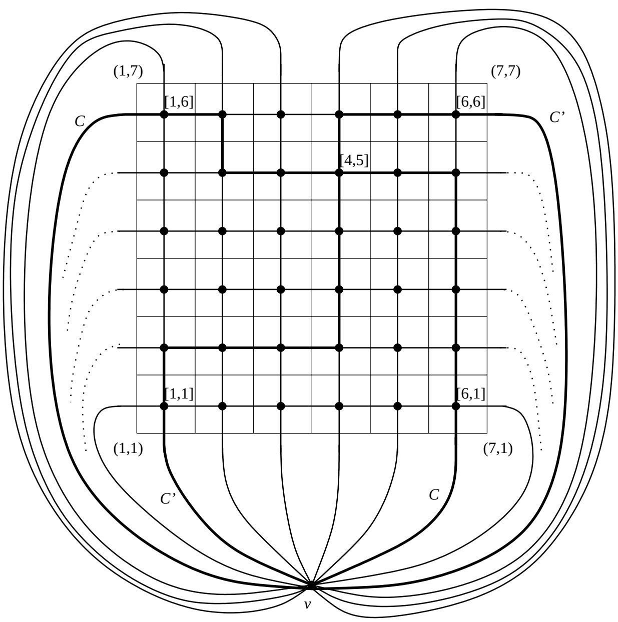


Fig. 3. Illustration of the dual graph argument

Proof. Let us denote $A = \{(n-2, i), i = 2, 3, \ldots, n-1\}, B = \{(j, n), j = 1, 2, \ldots, n-2\}, C = \{(i, 1), i = 1, 2, \ldots, n-2\}, D = \{(1, j), j = 2, 3, \ldots, n-1\}$. As in Proposition 1 we construct at least $2n-4$ edge disjoint paths between I and II. Let there are exactly $n-2$ edge disjoint paths between the black vertices of I and II. In this case the vertices from A are in say I and the vertices from B are in II.

If $C \subset I$ then we construct $n-2$ new edge disjoint paths between C and B: $(i, 1)(i, n)$ where $i = 1, 2, \ldots, n-2$. In total we have $3n-6$ edge disjoint paths and as $n \geq 3$ we are done.

If $C \subset II$ we construct one new edge disjoint path between C and A: : $(n-2, 1)(n-2, 2)$ and one new between D and B: $(1, n-1)(1, n)$. Totally there are $2n-2$ edge disjoint paths and we are done too.

If the vertices of C are partly in I and partly in II, according to the Proposition 1 we have at least $n-1$ edge disjoint paths between the white vertices of I and II. Hence there are together $2n-3$ edge disjoint paths between I and II and we are done. □

Proposition 6. *For* $n \geq 3$

$$ccw(P_{n-1} \times C_n) \geq n.$$

Proof. Let us denote $A = \{(n-1, i), i = 1, 2, \ldots, n-1\}, B = \{(j, n), j = 1, 2, \ldots, n-1\}, C = \{(i, 1), i = 1, 2, \ldots, n-2\}, D = \{(1, j), j = 2, 3, \ldots, n-1\}$.

There are at least $n-1$ and $n-2$ edge disjoint paths between black and white vertices of I and II, respectively. Let $A \subset I$ and $B \subset II$. We construct one new edge disjoint path between A and B: $(n-1,1)(n-1,n)$.
If $C \subset I$ then we construct new edge disjoint paths between C and B: $(i,1)(i,n)$ for $i = 1,2,\ldots,n-2$ and we are done because we have at least $2n-1$ edge disjoint paths between I and II.
If $C \subset II$ there are two other edge disjoint paths: one between D an B: $(1,n-1)(1,n)$ and one between C and A: $(n-2,1)(n-1,1)$.
If there exist a k such that $(1,n-1),(1,n-2),\ldots,(1,k) \in I$ (or II),
$(1,k-1),(1,k-2),\ldots,(1,1),(1,2),\ldots,(1,n-k) \in II$ (or I),
$(1,n-k+1),(1,n-k+2),\ldots,(1,n-2) \in I$ (or II) (this actually means that $C \cup D$ is cut into 3 subpaths) then we construct a new edge disjoint path between D and B: $(1,n-1)(1,n)$ (or between C and A: $(n-2,1)(n-1,1)$) and we are done.
Otherwise, according to the Proposition 1 there are at least n and $n-1$ edge disjoint paths between the black and white vertices of I and II, respectively and we are done. □

Proposition 7. *For $n \geq 3$*

$$ccw(P_n \times C_n) \geq n+1, ccw(P_{n+1} \times C_n) \geq n+2.$$

Proof. Left for the full version.

4 Conclusions

We created a new method to show exact results for the cyclic cutwidths of the 2-dimensional ordinary and cylindrical meshes. The method is applicable to mesh-like and product graphs. There are only a few classes of interconnection networks including meshes, complete graphs and trees, for which the cyclic cutwidth is known precisely. Cyclic cutwidth of the hypercube is an open problem. An interesting question is to characterize the class of interconnection networks for which the cutwidth and the cyclic cutwidth is the same.

Acknowledgment

The work of the second author was partly done while he was visiting LaBRI, Université Bordeaux I, team of Prof. A. Raspaud.

References

1. Alon, N., West, D. B., The Borsuk-Ulam theorem and bisection of necklaces, *Proceedings of the American Mathematical Society* **98** (1986), 623–628. 451

2. Bel Hala, A., Congestion optimale du plongement de l'ypercube $H(n)$ dans la chaine $P(2^n)$, *RAIRO Informatique, Théorique et Applications* **27** (1993), 1–17. 450
3. Bermond, J.-C., Gargano, L., Perennes, S., Rescigno, A., Vaccaro, U., Efficient collective communication in optical networks, in: *Proc. of ICALP'96,* LNCS 1099, Springer Verlag, Berlin, 1996, 574–585. 449
4. Bezrukov, S. L., Edge isoperimetric problems on graphs, in: *Proc. Bolyai Mathematical Studies*, Budapest, 1998. 449
5. Bezrukov, S. L., Schroeder, U.-P., The cyclic wirelength of trees, *Discrete Applied Mathematics* **87** (1998), 275–277. 450
6. Chavez, J. D., Trapp, R., The cyclic cutwidth of trees, *Discrete Applied Mathematics* **87** (1998), 25–32. 450
7. Chung, F. R. K., Labelings of graphs, in: *Selected Topics in Graph Theory 3*, (L. Beineke and R. Wilson eds.), Academic Press, New York, 1988, 151–168.
8. Gavril, F., Some $NP-$complete problems on graphs, in: *Proc. of the 11th Conf. on Information Sciences and Systems,* Johns Hopkins University, Baltimore, MD, 1977, 91–95. 450
9. Goldberg, C. H., West, D. B., Bisection of circle colorings, *SIAM J. Alg. Disc. Meth.* **6** (1985), 93–106. 451
10. Hu, Q., Zhang, Y., Shen, X., Rearrangeable graphs, in: *Proc. of COCOON'97*, LNCS 1276, Springer Verlag, Berlin, 1997, 441–450. 449
11. Hromkovič, J., Müller, V., Sýkora, O., Vrťo, I., On embeddings in cycles, *Information and Computation* **118** (1995), 302–305. 450
12. Lin, Y., Sýkora, O., Vrťo, I., On cyclic cutwidths, submitted for publication, 1997. 450, 450, 451
13. Lopez, A. D., Law, H. F. S., A dense gate matrix layout method for MOS VLSI, *IEEE Transactions on Electronic Devices* **27**, (1980), 1671–1675. 449
14. Paterson, M. S., Schröder, H., Sýkora, O., Vrťo, I., On permutation communications in all-optical rings, in: *Proc. of SIROCCO'98,* University of Salerno, Amalfi, 1998, 1–7. 449
15. Rolim, J., Sýkora, O., Vrťo, I., Optimal cutwidth of meshes, in: *Proc. 21th Intl. Workshop on Graph-Theoretic Concepts in Computer Science*, LNCS 1017, Springer Verlag, Berlin, 1995, 252–264. 450, 450, 450, 451
16. Schröder, H., Sýkora, O., Vrťo, I., Optical all-to-all communication for some product graphs, in: *Proc. of SOFSEM'97*, LNCS 1338, Springer Verlag, Berlin, 1997, 555–562. 449

Some Afterthoughts on Hopfield Networks

Jiří Šíma[1]*, Pekka Orponen[2], and Teemu Antti-Poika[2]**

[1] Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic,
sima@uivt.cas.cz

[2] Department of Mathematics, University of Jyväskylä,
P.O. Box 35, FIN-40351 Jyväskylä, Finland,
{orponen,anttipoi}@math.jyu.fi

Abstract. In the present paper we investigate four relatively independent issues, which complete our knowledge regarding the computational aspects of popular Hopfield nets. In Section 2 of the paper, the computational equivalence of convergent asymmetric and Hopfield nets is shown with respect to network size. In Section 3, the convergence time of Hopfield nets is analyzed in terms of bit representations. In Section 4, a polynomial time approximate algorithm for the minimum energy problem is shown. In Section 5, the Turing universality of analog Hopfield nets is studied.

1 Introduction

In his 1982 paper [12], John Hopfield introduced a very influential associative memory model which has since come to be known as the discrete-time Hopfield (or symmetric recurrent) network. Particularly, Hopfield nets compared with general asymmetric networks have favorable convergence properties. Part of the appeal of Hopfield nets also stems from their connection to the much-studied Ising spin glass model in statistical physics [3], and their natural hardware implementations using electrical networks [13] or optical computers [7]. Hopfield nets are suited for applications that require the capability to remove noise from large binary patterns. Besides associative memory, the proposed uses of Hopfield networks include, e.g., fast approximate solution of combinatorial optimization problems [14,32]. Although the practical applicability of Hopfield nets seems to be limited because of their low storage capacity, this fundamental model has inspired other important neural network architectures such as the BAM, Boltzmann machines, etc. [24]. Thus the theoretical analysis of Hopfield nets is also worthy for understanding the computational capabilities of the corresponding models.

We will first briefly specify the model of a finite *discrete recurrent neural network*. The network consists of n simple computational *units* or *neurons*, indexed

* Research supported by GA ČR Grant No. 201/98/0717.

** Research supported by Academy of Finland Grant No. 37115/96.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 459–469, 1999.

as $1, \ldots, n$, which are connected into a generally cyclic oriented graph or *architecture* in which each edge (i, j) leading from neuron i to j is labeled with an integer *weight* $w(i, j) = w_{ji}$. The absence of a connection within the architecture corresponds to a zero weight between the respective neurons. Special attention will be paid to *Hopfield* (*symmetric*) *networks*, whose architecture is an undirected graph with symmetric weights $w(i, j) = w(j, i)$ for every i, j.

We will mostly consider the *synchronous* computational dynamics of the network, working in *fully parallel mode*, which determines the evolution of the *network state* $\mathbf{y}^{(t)} = (y_1^{(t)}, \ldots, y_n^{(t)}) \in \{0, 1\}^n$ for all discrete time instants $t = 0, 1, \ldots$ as follows. At the beginning of the computation, the network is placed in an *initial state* $\mathbf{y}^{(0)}$ which may include an external input. At discrete time $t \geq 0$, each neuron $j = 1, \ldots, n$ collects its binary *inputs* from the *states* (*outputs*) $y_i^{(t)} \in \{0, 1\}$ of incident neurons i. Then its integer *excitation* $\xi_j^{(t)} = \sum_{i=0}^{n} w_{ji} y_i^{(t)}$ $(j = 1, \ldots, n)$ is computed as the respective weighted sum of inputs including an integer *bias* w_{j0} which can be viewed as the weight of the formal constant unit input $y_0^{(t)} = 1$, $t \geq 0$. At the next instant $t + 1$, an *activation function* σ is applied to $\xi_j^{(t)}$ for all neurons $j = 1, \ldots, n$ in order to determine the new network state $\mathbf{y}^{(t+1)}$ as follows:

$$y_j^{(t+1)} = \sigma\left(\xi_j^{(t)}\right) \quad j = 1, \ldots, n \tag{1}$$

where a *binary-state* neural network employs the *hard limiter* (or *threshold*) activation function

$$\sigma(\xi) = \begin{cases} 1 & \text{for } \xi \geq 0 \\ 0 & \text{for } \xi < 0\,. \end{cases} \tag{2}$$

Alternative computational dynamics are also possible in Hopfield nets. For example, under *sequential mode* only one neuron updates its state according to (1) at each time instant while the remaining neurons do not change their outputs. Or in Section 5 we will deal with the finite *analog-state* discrete-time recurrent neural networks which, instead of the threshold activation function (2), employ e.g. the *saturated-linear* sigmoid activation function

$$\sigma(\xi) = \begin{cases} 1 & \text{for } \xi > 1 \\ \xi & \text{for } 0 \leq \xi \leq 1 \\ 0 & \text{for } \xi < 0\,. \end{cases} \tag{3}$$

Hence the states of analog neurons are real numbers within the interval $[0, 1]$, and similarly the weights (including biases) are allowed to be reals.

The fundamental property of symmetric nets is that on their state space a bounded Liapunov, or 'energy' function can be defined, whose values are properly decreasing along any non-constant computation path (*productive* computation) of such a network. Specifically, given a sequential computation of a symmetric Hopfield net with, for simplicity, zero feedbacks $w_{jj} = 0$ and biases $w_{j0} = 0$, and, without loss of generality [22], non-zero excitations $\xi_j^{(t)} \neq 0$, $j = 1, \ldots, n$, one can associate an *energy* with state $y^{(t)}$, $t \geq 0$, as follows:

$$E\left(y^{(t)}\right) = E(t) = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ji} y_i^{(t)} y_j^{(t)} . \tag{4}$$

Hopfield [12] showed that for this energy function, $E(t) \leq E(t-1) - 1$ for every $t \geq 1$ of a productive computation. Moreover, the energy function is bounded, i.e. $|E(t)| \leq W$, where

$$W = \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} |w_{ji}| \tag{5}$$

is called the *weight* of the network. Hence, the computation must converge to a stable state within time $O(W)$. An analogous result can be shown for parallel updates, where a cycle of length at most two different states may appear [23].

The present paper discusses four relatively independent issues related to the computational properties of Hopfield networks [9], and presents new results concerning the computational equivalence of asymmetric and Hopfield networks, convergence time analysis, polynomial time approximate solution of the minimum energy problem, and the Turing universality of analog Hopfield nets. For lack of space, the proofs here are sketched or omitted; the full proofs can be found in the technical report [30].

2 A Size-Optimal Simulation of Asymmetric Networks

The computational power of symmetric Hopfield nets is properly less than that of asymmetric networks due to their different asymptotic behavior. Because of the Liapunov property, Hopfield networks always converge to a stable state (or a cycle of length two for parallel updates), whereas asymmetric networks can have limit cycles of arbitrary length. However, it is known [21] that this is the *only* feature that cannot be reproduced, in the sense that any *converging* fully parallel computation by a network of n discrete-time binary neurons, with in general asymmetric interconnections, can be simulated by a Hopfield net of size $O(n^2)$. More precisely, given an asymmetric network to be simulated, there exists a subset of neurons in the respective Hopfield net whose states correspond to the original convergent asymmetric computation in the course of the simulation — possibly with some constant time overhead per each original update. The idea behind this simulation is that each asymmetric edge to be simulated is implemented by a small symmetric subnetwork which receives "energy support" from a symmetric clock subnetwork (a binary counter) [11] in order to propagate a signal in the correct direction.

In the context of infinite families of neural networks, which contain one network for each input length (a similar model is used in the study of Boolean circuit complexity [33]), this simulation result implies that infinite sequences of discrete symmetric networks with a polynomially increasing number of neurons are computationally equivalent to (nonuniform) polynomially space-bounded Turing machines. Stated in standard complexity theoretic notation [33], such sequences of networks thus compute the complexity class PSPACE/poly, or P/poly when the weights in the networks are restricted to be polynomial in the input size [21].

In the following theorem the construction from [21] is improved by reducing the number of neurons in the simulating symmetric network from quadratic to *linear* as compared to the size of the asymmetric network; this result is asymptotically optimal. The improvement is achieved by simulating the individual neurons (as opposed to edges, as in [21]), whose states are updated by means of the clock technique. A similar idea was used for an analogous continuous-time simulation in [29].

Theorem 1. *Any fully parallel computation by a recurrent neural network of n binary neurons with asymmetric weights, converging within $t^\star$ discrete update steps, can be simulated by a Hopfield net with $6n+2$ neurons, converging within $4t^\star$ discrete-time steps.*

Proof. (Sketch) Observe, first, that any converging computation by an asymmetric network of n binary neurons must terminate within $t^\star \leq 2^n$ steps. A basic technique used in our proof is the exploitation of an $(n+1)$-bit symmetric clock subnetwork (a binary counter) which, using $3n+1$ units, produces a sequence $(0111)^{2^n}$ of 2^n well-controlled oscillations before it converges. This sequence of clock pulses generated by the least significant counter unit c_0 in the clock subnetwork is used to drive the rest of the network. The construction of the $(n+1)$-bit binary counter is omitted (see [30]). We only assume that the corresponding weights are chosen to be sufficiently large so that the clock is not influenced by the subnetwork it drives. In addition, a neuron $\bar{c}_0$ is added which computes the negation of c_0 output.

Then for each neuron j from the asymmetric network, 3 units p_j, q_j, r_j are introduced in the Hopfield net so that p_j represents the new (current) state $y_j^{(t)}$ of j at time $t \geq 1$ while q_j stores the old state $y_j^{(t-1)}$ of j from the preceding time instant $t-1$, and r_j is an auxiliary neuron realizing the update of the old state. In the sequel the symmetric weights in the Hopfield net will be denoted by w' whereas w denotes the original asymmetric weights. The corresponding symmetric subnetwork simulating one neuron j is depicted in Figure 1, where $U = \max_{j=1,\ldots,n} \sum_{i=0}^{n} |w_{ji}|$. Here the symmetric connections between neurons are labeled with the corresponding weights, and the biases are indicated by the edges drawn without an originating unit. The total number of units simulating the asymmetric network is $3n+1$ (including $\bar{c}_0$) which, together with the clock size $3n+1$, gives a total of $6n+2$ neurons in the Hopfield net.

At the beginning of the simulation all the neurons in the Hopfield net are passive (their states are zero), except for those units q_j corresponding to the original initially active neurons j, i.e. $y_j^{(0)} = 1$. Then an asymmetric network update at time $t \geq 1$ is simulated by a cycle of four steps in the Hopfield net as

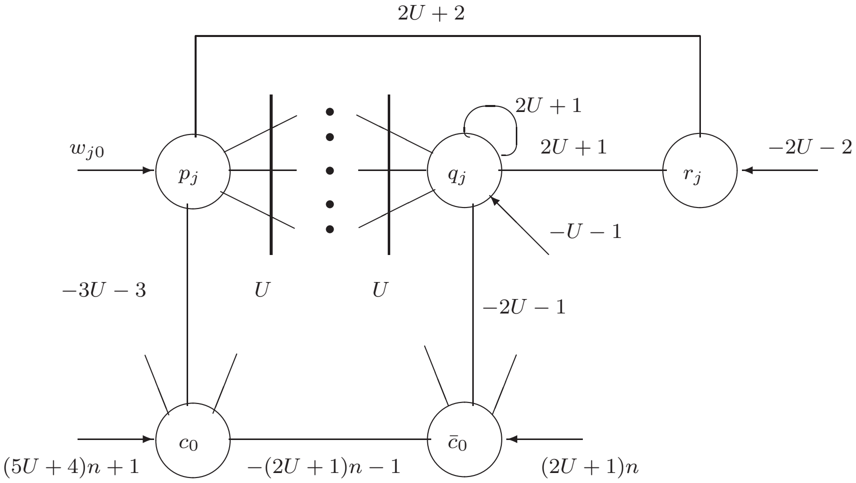


Fig. 1. Symmetric simulation of neuron j

follows. In the first step, unit c_0 fires and remains active until its state is changed by the clock since its large positive bias makes it independent of all the n neurons p_j. Also the unit $\bar{c}_0$ fires because it computes the negation of c_0 that was initially passive. At the same time each neuron p_j computes its new state $y_j^{(t)}$ from the old states $y_i^{(t-1)}$, which are stored in the corresponding units q_i. Thus each neuron p_j is connected with units q_i via the original weights $w'(q_i, p_j) = w(i, j)$ and also its bias $w'(0, p_j) = w(0, j)$ is preserved. So far, unit q_j keeps the old state $y_j^{(t-1)}$ due to its feedback. In the second step, the new state $y_j^{(t)}$ is copied from p_j to r_j, and the active neuron c_0 makes each neuron p_j passive by means of a large negative weight which exceeds the positive influence from units q_i ($i = 1, \ldots, n$) including its bias $w(0, p_j)$ according to the definition of U. Similarly, the active neuron $\bar{c}_0$ erases the old state $y_j^{(t-1)}$ from each neuron q_j by making it passive with the help of a large negative weight which, together with the negative bias, exceeds its feedback and the positive influence from units p_i ($i = 1, \ldots, n$). Finally, also neuron $\bar{c}_0$ becomes passive since c_0 was active. In the third step, the current state $y_j^{(t)}$ is copied from r_j to q_j since all the remaining incident neurons p_i and $\bar{c}_0$ are and remain passive due to c_0 being active. Therefore also unit r_j becomes passive. In the fourth step, c_0 becomes passive and the state $y_j^{(t)}$, being called old from now on, is stored in q_j. Thus the Hopfield net finds itself at the starting condition of the time $t+1$ asymmetric network simulation step, which proceeds in the same way. Altogether, the whole simulation is achieved within $4t^\star$ discrete-time steps. □

3 Convergence Time Analysis

In this section the *convergence time* in Hopfield networks, i.e. the number of discrete updates required before the network converges, will be analyzed. We shall consider only the worst case bounds; an average-case analysis can be found in [18]. Since a network with n binary neurons has 2^n different states, a trivial 2^n upper bound on the convergence time in symmetric networks of size n holds. On the other hand, the symmetric clock network [11] which is used in the proof of Theorem 1 provides an explicit example of a Hopfield net whose convergence time is exponential with respect to n. More precisely, this network yields a $\Omega(2^{n/3})$ lower bound on the convergence time of Hopfield nets, since a $(k+1)$-bit binary counter can be implemented using $n = 3k + 1$ neurons.

However, the above-mentioned bounds do not take into account the size of the *weights* in the network. An upper bound of $O(W)$ follows from the characteristics of the energy function (see Section 1), and this estimate can even be made more accurate by using a slightly different energy function [8]. This yields a polynomial upper bound on the convergence time of Hopfield nets with polynomial weights. Similar arguments can be used for fully parallel updates.

In the following theorem these results will be translated into convergence time bounds with respect to the length of *bit representations* of Hopfield nets. More precisely, for a symmetric network which is described within M bits, convergence-time lower and upper bounds $2^{\Omega(M^{1/3})}$ and $2^{O(M^{1/2})}$, respectively will be shown. It is an open problem whether these bounds can be improved. This is an important issue since the convergence-time results for binary-state networks can be compared with those for analog-state (or even continuous-time) networks in which the precision of real weight parameters (i.e. the representation length) plays an important role. For example, there exists an analog-state symmetric network with an encoding size of M bits that converges after $2^{\Omega(g(M))}$ continuous-time units, where $g(M)$ is an arbitrary continuous function such that $g(M) = o(M)$, $g(M) = \Omega(M^{2/3})$, and $M/g(M)$ is increasing [29]. From the result presented here it follows that the computation of this analog symmetric network terminates later than that of any other discrete Hopfield net of the same representation size. Note also that in this approach we express the convergence time with respect to the *full* descriptional complexity of the Hopfield net, not just the number of neurons which captures its computational resources only partially.

Theorem 2. *There exists a Hopfield network with an encoding size of M bits that converges after $2^{\Omega(M^{1/3})}$ updates. On the other hand, any computation of a symmetric network with a binary representation of M bits terminates within $2^{O(M^{1/2})}$ discrete computational steps.*

Proof. (Sketch) For the underlying lower bound the clock network from the proof of Theorem 1 can again be exploited. For the upper bound, consider a Hopfield network with an M-bit representation that converges after $T(M)$ updates. A major part of this M-bit representation consists of m binary encodings of weights

$w_1, \dots, w_m$ of the corresponding lengths $M_1, \dots, M_m$ where $\sum_{r=1}^{m} M_r = \Theta(M)$. Clearly, there must be at least $T(M)$ different energy levels corresponding to the states visited during the computation. Thus the underlying weights must produce at least $S \geq T(M)$ different sums $\sum_{r \in A} w_r$ for $A \subseteq \{1, \dots, m\}$ where w_r for $r \in A$ agrees with w_{ji} for $y_i = y_j = 1$ in (4). So, it is sufficient to upper bound the number of different sums over m weights whose binary representations form a $\Theta(M)$-bit string altogether. This yields $T(M) \leq 2^{O(M^{1/2})}$. □

4 Approximating the Minimum Energy Problem

Another important issue in Hopfield nets is the *MIN ENERGY* or *GROUND STATE* problem of finding a network state with minimal energy (4) for a given symmetric neural network. Remember that in (4) it is assumed, for reasons of simplicity, that $w_{jj} = 0$ and $w_{j0} = 0$ for $j = 1, \dots, n$. In addition, without loss of generality [22], we shall work throughout this section with frequently used bipolar states $-1, 1$ of neurons instead of binary ones $0, 1$. This problem appears to be of a special interest since many hard combinatorial optimization problems have been heuristically solved by minimizing the energy in Hopfield nets [1,14]. This issue is also important in the Ising spin glass model in statistical physics [3].

Unfortunately, the decision version of the MIN ENERGY problem, i.e. whether there exists a network state having an energy less than a prescribed value, is NP-complete. This can be observed from the above-mentioned reductions of hard optimization problems to MIN ENERGY. For an explicit NP-completeness proof see e.g. [34] where a reduction from SAT is exploited. On the other hand, MIN ENERGY has polynomial time algorithms for Hopfield nets whose architectures are planar lattices [6] or planar graphs [3].

Perhaps the most direct and frequently used reduction to MIN ENERGY is from the *MAX CUT* problem (see e.g. [4]). In MAX CUT we are given an undirected graph $G = (V, E)$ with an integer edge cost function $c \colon E \longrightarrow \mathcal{Z}$, and we want to find a *cut* $V_1 \subseteq V$ which maximizes the *cut size*

$$c(V_1) = \sum_{\{i,j\} \in E, i \in V_1, j \notin V_1} c(\{i,j\}) - \sum_{\{i,j\} \in E, c(\{i,j\}) < 0} c(\{i,j\}) \,. \tag{6}$$

Note that the standard MAX CUT problem is here generalized by allowing also the negative edge weights that are needed for the opposite reduction from MIN ENERGY to MAX CUT. Recently, a new randomized approximation algorithm with a high performance guarantee $\alpha = 0.87856$ for this MAX CUT formulation has been proposed [10] and later de-randomized [20]: a fact which we will exploit for approximating the MIN ENERGY problem. Namely, we will observe that MIN ENERGY can be approximated in a polynomial time within absolute error less than $0.243W$ where W is the network weight (5). For $W = O(n^2)$, e.g. for Hopfield nets with n neurons and constant weights, this result matches the lower bound $\Omega(n^{2-\varepsilon})$ which cannot be guaranteed by any approximate polynomial time MIN ENERGY algorithm for every $\varepsilon > 0$ [4], unless $P = NP$. In addition, an approximate polynomial time MIN ENERGY algorithm with absolute error

$O(n/\log n)$ is also known in a special case of Hopfield nets whose architectures are two-level grids [5].

Theorem 3. *The MIN ENERGY problem for Hopfield nets can be approximated in a polynomial time within the absolute error less than* $0.243W$ *where* W *is the network weight (5).*

Proof. (Sketch) We first recall the well-known simple reduction between MIN ENERGY and MAX CUT problems. For a Hopfield network with architecture G and weights $w(i,j)$ we can easily define the corresponding instance $G = (V,E); c$ of MAX CUT with edge costs $c(\{i,j\}) = -w(i,j)$ for $\{i,j\} \in E$. It can be shown that any cut $V_1 \subseteq V$ of G corresponds to a Hopfield net state $\mathbf{y} \in \{-1,1\}^n$ where $y_i = 1$ if $i \in V_1$ and $y_i = -1$ for $i \in V \setminus V_1$, so that the respective cut size $c(V_1)$ is related to the underlying energy $E(\mathbf{y}) = W - 2c(V_1)$. This implies that the minimum energy state corresponds to the maximum cut.

Now, the approximate polynomial time algorithm from [10] can be employed to solve instance $G = (V,E); c$ of the MAX CUT problem which provides a cut V_1 whose size $c(V_1) \geq \alpha c^\star$ is guaranteed to be at least $\alpha = 0.87856$ times the maximum cut size $c^\star$. Let cut V_1 correspond to the Hopfield network state $\mathbf{y}$ which implies $c(V_1) = 1/2(W - E(\mathbf{y}))$. Hence, we get a guarantee $W - E(\mathbf{y}) \geq \alpha(W - E^\star)$ where $E^\star$ is the minimum energy corresponding to the maximum cut $c^\star$ which leads to $E(\mathbf{y}) - E^\star \leq (1-\alpha)(W - E^\star)$. Since $|E^\star| \leq W$, we obtain the desired guarantee for the absolute error $E(\mathbf{y}) - E^\star \leq (1-\alpha)2W < 0.243W$. □

5 Turing Universality of Finite Analog Hopfield Nets

In this section we deal with the computational power of finite *analog-state* discrete-time recurrent neural networks. For asymmetric analog networks, the computational power is known to increase with the Kolmogorov complexity of their real weights [2]. With integer weights such networks are equivalent to finite automata [15,16,31], while with rational weights arbitrary Turing machines can be simulated [16,26]. With arbitrary real weights the network can even have 'super-Turing' computational capabilities, e.g. polynomial time computations correspond to the complexity class P/poly and all languages can be recognized within exponential time [25]. On the other hand, any amount of analog noise reduces the computational power of this model to that of finite automata [19].

For finite symmetric networks, only the computational power of binary-state Hopfield nets is fully characterized. Namely, they recognize the so-called Hopfield languages [27], which form a proper subclass of regular languages; hence such networks are less powerful than finite automata. Hopfield languages can also be faithfully recognized by analog symmetric neural networks [19,28] and this provides the lower bound on their computational power. A natural question arises whether finite analog Hopfield nets are Turing universal, i.e. whether a Turing machine simulation can be achieved with rational weights similarly as in the asymmetric case [16,26]. The main problem is that under fully parallel updates any analog Hopfield net with rational weights converges to a limit cycle of length at most two [17]. Thus the only possibility of simulating Turing machines would be to exploit a sequence of rational network states converging to this limit cycle which seems to be tricky if possible at all. A more reasonable approach is to supply an external clock that produces an infinite sequence of binary pulses providing the symmetric network with an energy support, e.g. for simulating an asymmetric analog network similarly as in Theorem 1. In this way the computational power of analog Hopfield nets with an external clock is proved to be the same as that of asymmetric analog networks. Especially for rational weights, this implies that such networks are Turing universal. The following theorem also completely characterizes those infinite binary sequences by the external clock which prevent the Hopfield network from converging.

Theorem 4. *Let N be an analog-state recurrent neural network with real asymmetric weights and n neurons working in fully parallel mode. Then there exists an analog Hopfield net N' with $3n + 8$ units whose real weights have the same maximum Kolmogorov complexity as the weights in N, and which simulates N, provided it receives as an additional external input any infinite binary sequence that contains infinitely many substrings of the form $bx\bar{b} \in \{0,1\}^3$ where $b \neq \bar{b}$. Moreover, the external input must have this property to prevent N' from converging.*

References

1. Aarts, E., Korst, J. *Simulated Annealing and Boltzmann Machines.* Wiley and Sons, 1989. 465
2. Balcázar, J. L., Gavaldà, R., Siegelmann, H. T. Computational power of neural networks: A characterization in terms of Kolmogorov complexity. *IEEE Transactions of Information Theory*, **43**, 1175–1183, 1997. 466
3. Barahona, F. On the computational complexity of Ising spin glass models. *Journal of Physics A*, **15**, 3241–3253, 1982. 459, 465, 465
4. Bertoni, A., Compadelli, P. On the approximability of the energy function. In *Proceedings of the ICANN'94 conference*, 1157–1160, Springer-Verlag, 1994. 465, 465
5. Bertoni, A., Campadelli, P., Gangai, C., Posenato, R. Approximability of the ground state problem for certain Ising spin glasses. *Journal of Complexity*, **13**, 323–339, 1997. 466
6. Bieche, I., Maynard, R., Rammal, R., Uhry, J. P. On the ground states of the frustration model of a spin glass by a matching method of graph theory. *Journal of Physics A*, **13**, 2553–2576, 1980. 465
7. Farhat, N. H., Psaltis, D., Prata, A., Paek, E. Optical implementation of the Hopfield model. *Applied Optics*, **24**, 1469–1475, 1985. 459
8. Floreén, P. Worst-case convergence times for Hopfield memories. *IEEE Transactions on Neural Networks*, **2**, 533–535, 1991. 464
9. Floreén, P., Orponen, P. Complexity issues in discrete Hopfield networks. Research Report A–1994-4, Department of Computer Science, University of Helsinki, 1994. 461
10. Goemans, M. X., Williamson, D. P. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, **42**, 1115–1145, 1995. 465, 466

11. Goles, E., Martínez, S. Exponential transient classes of symmetric neural networks for synchronous and sequential updating. *Complex Systems*, **3**, 589–597, 1989. 461, 464
12. Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*, vol. **79**, 2554–2558, 1982. 459, 461
13. Hopfield, J. J. Neurons with graded response have collective computational properties like those of two-state neurons. In *Proceedings of the National Academy of Sciences*, vol. **81**, 3088–3092, 1984. 459
14. Hopfield, J. J., Tank, D. W. "Neural" computation of decisions in optimization problems. *Biological Cybernetics*, **52**, 141-152, 1985. 459, 465
15. Horne, B. G., Hush, D. R. Bounds on the complexity of recurrent neural network implementations of finite state machines. *Neural Networks*, **9**, 243–252, 1996. 466
16. Indyk, P. Optimal simulation of automata by neural nets. In *Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science*, vol. **900** of LNCS, 337-348, Springer-Verlag, 1995. 466, 466, 466
17. Koiran, P. Dynamics of discrete time, continuous state Hopfield networks. *Neural Computation*, **6**, 459–468, 1994. 467
18. Komlós, P., Paturi, R. Convergence results in an associative memory model. *Neural Networks*, **1**, 239–250, 1988. 464
19. Maass, W., Orponen, P. On the effect of analog noise in discrete-time analog computations. *Neural Computation*, **10**, 1071–1095, 1998. 466, 466
20. Mahajan, S., Ramesh, H. Derandomizing semidefinite programming based approximation algorithms. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science.*, 162–163, IEEE, Los Alamitos, California, 1995. 465
21. Orponen, P. The computational power of discrete Hopfield nets with hidden units. *Neural Computation*, **8**, 403–415, 1996. 461, 462, 462, 462
22. Parberry, I. A primer on the complexity theory of neural networks. In *Formal Techniques in Artificial Intelligence: A Sourcebook*, editor R. B. Banerji, 217–268, Elsevier, North-Holland, 1990. 461, 465
23. Poljak, S., Súra, M. On periodical behaviour in societies with symmetric influences. *Combinatorica*, **3**, 119–121, 1983. 461
24. Rojas, R. *Neural Networks: A Systematic Introduction.* Springer Verlag, 1996. 459
25. Siegelmann, H. T., Sontag, E. D. Analog computation via neural networks. *Theoretical Computer Science*, **131**, 331–360, 1994. 466
26. Siegelmann, H. T., Sontag, E. D. Computational power of neural networks. *Journal of Computer System Science*, **50**, 132–150, 1995. 466, 466
27. Šíma, J. Hopfield languages. In *Proceedings of the SOFSEM Seminar on Current Trends in Theory and Practice of Informatics*, LNCS **1012**, 461–468, Springer-Verlag, 1995. 466
28. Šíma, J. Analog stable simulation of discrete neural networks. *Neural Network World*, **7**, 679–686, 1997. 466
29. Šíma, J., Orponen, P. A continuous-time Hopfield net simulation of discrete neural networks. Technical report V-773, ICS ASCR, Prague, January, 1999. 462, 464
30. Šíma, J., Orponen, P., Antti-Poika, T. Some afterthoughts on Hopfield networks. Technical report V-781, ICS ASCR, Prague, May, 1999. 461, 462
31. Šíma, J., Wiedermann, J. Theory of neuromata. *Journal of the ACM*, **45**, 155–178, 1998. 466
32. Sheu, B., Lee, B., Chang, C.-F. Hardware annealing for fast-retrieval of optimal solutions in Hopfield neural networks. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, Seattle, Vol. **II**, 327–332, 1991. 459

33. Wegener, I. *The Complexity of Boolean Functions.* Wiley/Teubner, 1987. 462, 462

34. Wiedermann, J. Complexity issues in discrete neurocomputing. *Neural Network World*, **4**, 99–119, 1994. 465

A Persistent-Set Approach to Abstract State-Space Construction in Verification

Ulrich Ultes-Nitsche[⋆]

Department of Electronics and Computer Science, University of Southampton
Southampton, SO17 1BJ, United Kingdom
un@ecs.soton.ac.uk

Abstract. When analysing a concurrent program in order to verify its correctness, in practice, one faces severe complexity problems. To cope with the problem of state-space explosion, two different types of approaches have been established: *abstraction techniques* and *partial-order methods* that both have certain drawbacks. To overcome these drawbacks we show in this paper how to combine abstraction with the "*persistent-set selective search*" partial-order method.

1 Introduction

In practice, the size of automata representations of the behaviour of concurrent programs limits the application of automatic verification concepts to rather small examples. State-space reduction can help to improve the efficiency of verification algorithms. Basically two main concepts for the construction of a reduced state-space exist: *abstraction techniques* [2,7,8] and *partial-order methods* [5,11,12]. Abstraction reduces the state-space by reducing the diversity of actions: actions can be ignored or identified with one another. Partial-order methods avoid considering particular interleavings of concurrent behavioural patterns. Though being useful for some examples, both concepts still have certain drawbacks related to their expressiveness and efficiency. In the present paper, we adapt the concept of an *abstraction-compatible* reduced state-space [10] to fit to *persistent-set selective search* [5,12]. This fills the gap in [10], where a *practical* result is lacking. We will show that a persistent-set reduction is suitable for computing abstractions as well as for checking requirements that ensure an abstraction to be sufficiently meaningful.

2 Preliminaries

The *behaviour* of a concurrent program can be represented by a set of infinitely long sequences of *actions*. We consider the set Σ of actions to be finite. Thus a behaviour is an ω-language on a finite set Σ of actions. We call each infinite sequence of actions a *computation* of the program. The sequences of actions that

[⋆] Former name: Ulrich Nitsche

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 470–478, 1999.

the program can perform in a finite amount of time are its *partial computations.* The *partial behaviour* is the set of all finitely long sequences of actions, i.e. it is a language on the set of actions. Since all prefixes of a partial computation are also partial computations of the program, we require that a partial behaviour is a *prefix-closed* language (subsequently let Σ^* be the set of all finitely long sequences on Σ and Σ^ω the set of all infinitely long sequences): Let $L \subseteq \Sigma^*$ and let $pre(L)$ designate the set of all prefixes of words in L. L is prefix-closed if and only if $pre(L) = L$.

The (infinite) behaviour of a concurrent program is determined by its partial behaviour continued to infinity. The idea of the infinite "continuation" of a partial behaviour can be defined using the *Eilenberg-limit* of a language [4]: The Eilenberg-limit $lim(L)$ of L is defined as $lim(L) = \{x \in \Sigma^\omega \mid |pre(x) \cap L| = \infty\}$.

We consider only *regular* behaviours in this paper, i.e. behaviours that can be represented by a finite automaton. We thus define: A behaviour of a concurrent program is the Eilenberg-limit $lim(L)$ of a prefix-closed regular language $L \subseteq \Sigma^*$. It is important to note that only dealing with languages is not a restriction since we can encode finite automata *completely* by their local languages [4] (the languages over transition triples $(state, event, sucessorstate)$).

A behaviour satisfies a *linear property* [1] if and only if all its computations satisfy it. So a property can be characterised by a set of (correct) computations [1]: A property $\mathcal{P}$ over Σ is a subset of Σ^ω. A behaviour $lim(L)$ satisfies $\mathcal{P}$ if and only if $lim(L) \subseteq \mathcal{P}$. (We write: "$lim(L) \models \mathcal{P}$".) To introduce an implicit fairness assumption into the satisfaction relation, we define *relative liveness properties* [6,8] of behaviours as an *approximate satisfaction* relation of properties [7]. To do so, we first have to introduce the notion of a language's leftquotient: The leftquotients of L and $lim(L)$ by the word w are defined as $cont(w, L) = \{v \in \Sigma^* \mid wv \in L\}$ and $cont(w, lim(L)) = \{x \in \Sigma^\omega \mid wx \in lim(L)\}$. A language's leftquotients describe exactly the states of the minimal automaton accepting the language.

We call $\mathcal{P} \subseteq \Sigma^\omega$ a relative liveness property of $lim(L)$ if and only if $\forall w \in pre(lim(L))\colon \exists x \in cont(w, lim(L))\colon wx \in \mathcal{P}$. If $lim(L) = \Sigma^\omega$, then the definitions of a relative liveness property is equivalent to the definition of a liveness property [1]. Usually, relative liveness is used to classify properties with respect to other properties [6]. In contrast to this, we consider relative liveness as a satisfaction relation with an inherent fairness condition that we call *approximate satisfaction* [7]: If $\mathcal{P} \subseteq \Sigma^\omega$ is a relative liveness property of $lim(L)$ we say that $lim(L)$ satisfies $\mathcal{P}$ approximately. We write "$lim(L) \models_{RL} \mathcal{P}$" to indicate that $lim(L)$ satisfies property $\mathcal{P}$ approximately. The name *approximate* satisfaction is motivated by observing that $\mathcal{P}$ is a relative liveness property of $lim(L)$ if and only if $lim(L) \cap \mathcal{P}$ is a dense set in the Cantor topology on $lim(L)$ [7]. In this sense approximate satisfaction is indeed an approximation to linear property satisfaction. We can also give an alternative characterisation on prefix sets: $lim(L)$ satisfies $\mathcal{P}$ approximately if and only if $pre(lim(L)) = pre(lim(L) \cap \mathcal{P})$ [7,8]. One can show that a finite-state implementation of a behaviour always exists such that all strongly fair computations of the implementation satisfy the property linearly [8]. This finite-state implementation can be much larger (the product of the state-spaces of the behaviour and the property) than the minimal finite-state representation of $lim(L)$. Therefore, approximate satisfaction can be viewed as satisfaction under fairness that allows for a very compact representation of a system's behaviour.

3 Abstraction

The size of a finite-state representation of the behaviour (the state-space) of a program limits the applicability of automatic verification techniques. On the other hand, a program usually performs actions which need not be considered in the verification process or which need not be distinguished from other actions respectively. We can reduce the state-space by ignoring unimportant actions or by giving a common name to actions which need not be distinguished from one another. These two concepts are *action hiding* and *action renaming* respectively. The so defined type of simplification is called *(behaviour) abstraction* [7,8]: Let Σ^∞ designate $\Sigma^* \cup \Sigma^\omega$. We call a mapping $h\colon \Sigma^\infty \to \Sigma'^\infty$ an *abstraction homomorphism* if and only if the following conditions hold:

- If we constrain h to a mapping on letters in Σ, then we obtain a *total* function $h\colon \Sigma \to \Sigma' \cup \{\varepsilon\}$ (action renaming and hiding).
- If $v, w \in \Sigma^*$ and $x \in \Sigma^\omega$, then $h(vw) = h(v)h(w)$ and $h(vx) = h(v)h(x)$ (compatibility with concatenation).
- If we constrain h to a mapping on ω-words over Σ, then we obtain a *partial* function $h\colon \Sigma^\omega \to \Sigma'^\omega$. (no reduction of infinite sequences to finite ones)

Note that abstraction homomorphisms are partial mappings since they are not defined on ω-words that would be taken to finitely long words. Since we are considering behaviours that are Eilenberg-limits of prefix-closed regular languages, we have to define abstractions in terms of Eilenberg-limits: We define the abstraction of the concrete behaviour $lim(L)$ to be $lim(h(L))$. This definition is reasonable since, for prefix-closed regular L, $lim(h(L))$ and $h(lim(L))$ are equal [8]. When considering approximately satisfied properties on an abstract behaviour, abstraction homomorphisms do not establish a suitable abstraction concept since they do not preserve approximately satisfied properties. *Preservation of properties* designates that a property which holds for the abstraction holds for the concrete behaviour in a corresponding way. To ensure preservation of approximately satisfied properties, an additional requirement must be satisfied by the abstraction homomorphism which is called *simplicity* of the homomorphism on the behaviour [9]: h is called *simple* on L if and only if, for all $w \in \Sigma^*$ and all $v \in cont(h(w), h(L))$ we have $cont(v, cont(h(w), h(L))) = cont(v, h(cont(w, L)))$. One should not worry about this fairly technical definition. Its content is best revealed by the following Theorem [7]:

Theorem 1. *If $h(L)$ does not contain maximal words*[1]*, then the condition*

$$lim(h(L)) \models_{RL} \mathcal{P} \quad \textit{if and only if} \quad lim(L) \models_{RL} h^{-1}(\mathcal{P})$$

holds if and only if h is simple on L.

According to this Theorem, we can derive approximately satisfied properties of a concurrent program from approximately satisfied properties of its abstraction if and only if the abstraction homomorphism is simple. In the subsequent sections we will show that the abstraction $lim(h(L))$ can be computed without constructing the concrete behaviour $lim(L)$ of the program exhaustively.

4 Partial-Order Methods and Abstraction

A different way to tackle the state-space explosion problem is established by *partial-order methods*. If there are several different partial computations of a program that are equal except for permutations of adjacent *independent* actions, only a reduced number of representatives is considered. Partial-order methods ignore interleavings of particular concurrent behavioural patterns. The key notion is the *independence* of actions: A relation $\Delta \subseteq \Sigma \times \Sigma$ is called an independence relation for $L \subseteq \Sigma^*$ if and only if it holds that $(a, a') \in \Delta$ if and only if for all $w \in L$ we have:

- if $a \in cont(w, L)$, then $a' \in cont(wa, L)$ if and only if $a' \in cont(w, L)$.
- if $aa' \in cont(w, L)$, then $cont(waa', L) = cont(wa'a, L)$.

We consider two partial computations w and w' to be equivalent (written: $w \equiv_\Delta w'$) if and only if we can transform one into the other by (repeatedly) permuting adjacent independent actions. The equivalence classes $[w]_\Delta$ of this equivalence relation are called *traces* [3]. Not surprisingly, partial computations that belong to the same trace lead to the same state (represented by leftquotients).

Lemma 1. *Let $w, w' \in L$ such that $w \equiv_\Delta w'$. Then $cont(w, L) = cont(w', L)$.*

Proof. Let $(w^i)_{1 \leq i \leq n}$, $n \in I\!N$, be a sequence of partial computations such that $w^1 = w$, $w^n = w'$, and for all $1 \leq i \leq n-1$, w^{i+1} can be derived from w^i by permuting exactly one pair of adjacent independent actions.

Let $w^i = u \cdot a \cdot b \cdot v$ and $w^{i+1} = u \cdot b \cdot a \cdot v$ such that $u, v \in \Sigma^*$, $a, b \in \Sigma$, and $(a, b) \in \Delta$. Because a and b are independent, $cont(uab, L) = cont(uba, L)$ and hence $cont(uabv, L) = cont(ubav, L)$. Therefore $cont(w^i, L) = cont(w^j, L)$, for all $1 \leq i, j \leq n$. □

[1] $w \in L$ is a maximal word in L if and only if $cont(w, L) = \{\varepsilon\}$. The restriction to maximal-word-free languages in the Theorem can be removed easily by introducing additional "dummy" actions to continue maximal words [7]. For the sake of not introducing additional notation, this obvious construction is omitted here.

We are now going to combine abstraction with partial-order methods in a suitable way, aiming to compute abstractions from partial-order representations rather than from exhaustive representations of a program's behaviour. We first define when an independence relation is compatible with an abstraction homomorphism: An independence relation $\Delta \subseteq \Sigma \times \Sigma$ is h-compatible if and only if $(a, b) \in \Delta$ implies $h(a) = h(b)$. We first prove that for h-compatible independence relations, equivalent partial computations have the same abstraction.

Lemma 2. *Let $\Delta \subseteq \Sigma \times \Sigma$ be an h-compatible independence relation. Let $w' \in \Sigma^*$ such that $w \equiv_\Delta w'$. Then $h(w) = h(w')$.*

Proof. Let $(w^i)_{1 \leq i \leq n}$, $n \in I\!N$, be a sequence of partial computations such that $w^1 = w$, $w^n = w'$, and for all $1 \leq i \leq n-1$, w^{i+1} can be derived from w^i by permuting exactly one pair of adjacent independent actions. Let $(a, b) \in \Delta$ be such a pair. Because Δ is h-compatible, we have $h(a) = h(b)$. Hence, $h(ab) = h(ba)$ and consequently $h(w^i) = h(w^{i+1})$, for all $1 \leq i \leq n-1$. Thus $h(w) = h(w')$. □

Let $\Delta \subseteq \Sigma \times \Sigma$ be an h-compatible independence relation. An h-compatible partial-order reduction $R^L_\Delta \subseteq \Sigma^*$ of L with respect to Δ is a prefix-closed language such that for all $w' \in R^L_\Delta$:

1. $\mathit{cont}(w', R^L_\Delta) \subseteq \mathit{cont}(w', L)$.
2. For all v in $\mathit{cont}(w', L)$ there exists v' in $\mathit{cont}(w', R^L_\Delta)$ such that $h(v) = h(v')$ and $h(\mathit{cont}(w'v, L)) = h(\mathit{cont}(w'v', R^L_\Delta))$.

The first part of this definition guarantees that R^L_Δ does not represent more than L does. The second part guarantees that R^L_Δ contains sufficient information about L to compute meaningful abstractions. Subsequently let $\Delta \subseteq \Sigma \times \Sigma$ always be an h-compatible independence relation, and let $w' \in R^L_\Delta$ and $w \in L$ such that $w \equiv_\Delta w'$.

Lemma 3. $h(\mathit{cont}(w, L)) = h(\mathit{cont}(w', R^L_\Delta))$.

Proof. Because $R^L_\Delta \subseteq L$ and $h(\mathit{cont}(w, L)) \subseteq h(\mathit{cont}(w', R^L_\Delta))$, the lemma follows immediately from the definition of an h-compatible trace system. □

Corollary 1. $h(L) = h(R^L_\Delta)$.

Proof. Let $w = w' = \varepsilon$. Then the corollary follows from Lemma 3. □

We can use the previous results to show that simplicity of an abstraction on the behaviour of a concurrent program can be checked already on its abstraction-compatible partial-order reduction.

Theorem 2. *h is simple on L if and only if h is simple on R^L_Δ.*

Proof. "$\Rightarrow$": Assume that h is simple on L. Let w' be in R^L_Δ. By definition, w' is in L. Since h is simple on L, there exists $v \in \mathit{cont}(h(w'), h(L))$ such that the set $\mathit{cont}(v, \mathit{cont}(h(w'), h(L)))$ is equal to $\mathit{cont}(v, h(\mathit{cont}(w', L)))$. By Lemma 3 and Corollary 1, $h(L) = h(R^L_\Delta)$ as well as $h(\mathit{cont}(w', L)) = h(\mathit{cont}(w', R^L_\Delta))$. So $\mathit{cont}(v, \mathit{cont}(h(w'), h(R^L_\Delta)))$ is equal to $\mathit{cont}(v, h(\mathit{cont}(w', R^L_\Delta)))$. Conseqently, h is simple on R^L_Δ.

"$\Leftarrow$": Assume that h is simple on R^L_Δ. Let w be in L. By definition, $w' \in R^L_\Delta$ exists such that $h(w) = h(w')$ and $h(\mathit{cont}(w, L)) = h(\mathit{cont}(w', R^L_\Delta))$. Since h is simple on R^L_Δ, there is $v' \in \mathit{cont}(h(w'), h(R^L_\Delta))$ so that $\mathit{cont}(v', h(\mathit{cont}(w', R^L_\Delta)))$ is equal to $\mathit{cont}(v', \mathit{cont}(h(w'), h(R^L_\Delta)))$. By Lemma 3, $h(L) = h(R^L_\Delta)$ as well as $h(\mathit{cont}(w, L)) = h(\mathit{cont}(w', R^L_\Delta))$. Therefore $\mathit{cont}(v', \mathit{cont}(h(w), h(L)))$ is equal to $\mathit{cont}(v', h(\mathit{cont}(w, L)))$ and thus h is simple on L. □

We can combine this result with the preservation result for approximately satisfied properties (Theorem 1) and Lemma 3 and obtain finally:

Corollary 2. *If $h(R^L_\Delta)$ does not contain maximal words, then the condition*

$$\mathit{lim}(h(R^L_\Delta)) \models_{RL} \mathcal{P} \quad \textit{if and only if} \quad \mathit{lim}(L) \models_{RL} h^{-1}(\mathcal{P})$$

holds if and only if h is simple on R^L_Δ.

5 Persistent Sets

The definition of a partial-order reduction in the previous section leaves it open how the reduction can be constructed effectively. We show in this section that a persistent-set reduction technique [5,12] allows for a practically feasible construction of an partial-order reduced state-space. The set $A \subseteq \mathit{cont}(w, L) \cap \Sigma$ is persistent in $\mathit{cont}(w, L)$ if an only if, for all actions a that occur in words in $(\Sigma - A)^* \cap \mathit{cont}(w, L)$, action a is independent of all actions in A. In other words, a set A of actions that are "active" in the state reached by w is called persistent if and only if all actions that are "reachable" by action sequences outside of A are independent of the actions in A. A persistent-set reduction to L is a language R_L in which for all $w \in R_L$, $\mathit{cont}(w, R_L) \cap \Sigma$ is a persistent set in $\mathit{cont}(w, L)$. Subsequently let $w' \in R_L$.

Lemma 4. *$\mathit{cont}(w', R_L) \subseteq \mathit{cont}(w', L)$.*

Proof. Assume $v' \in \mathit{cont}(w', R_L)$ exists that is not in $\mathit{cont}(w', L)$. Let u' be the longest common prefix of v' and a word in $\mathit{cont}(w', L)$ and let $a' \in \Sigma$ be the action that follows u' in v', i.e. $u'a' \in \mathit{pre}(v')$. By the choice of u', $u'a' \notin \mathit{cont}(w', L)$. So a' which is in $\mathit{cont}(w'u', R_L) \cap \Sigma$ is not in $\mathit{cont}(w'u', L) \cap \Sigma$. Therefore a' cannot be in a persistent set in $\mathit{cont}(w'u', L)$ which contradicts that R_L is a persistent-set reduction to L. □

Lemma 5. *$h(\mathit{cont}(w', L)) = h(\mathit{cont}(w', R_L))$.*

Proof. By Lemma 4 we only have to prove $h(cont(w', L)) \subseteq h(cont(w', R_L))$. We show this by induction on the length of $v \in cont(w', L)$. The base case is immediate since ε is in $cont(w', L)$ as well as in $cont(w', R_L)$.

If $|v| > 0$ then $v = a_1 a_2 a_3 \dots a_n$ for actions $a_i \in \Sigma$. Assume that all a_i are not in the persistent set in $cont(w', L)$. Let $p \in \Sigma$ be in this persistent set. Then $vp \in cont(w', L)$. Action p is independent to all a_i and therefore pv is also in $cont(w', L)$ as well as its prefix $pa_1 a_2 \dots a_{n-1}$. Because p can only be independent of a_i if $h(p) = h(a_i)$, $h(vp) = h(pv)$ which implies that $h(pa_1 a_2 \dots a_{n-1}) = h(v)$.

If some a_i are in the persistent set in $cont(w', L)$ then let j be the position of the first action in v that is in the persistent set in $cont(w', L)$. Let $p = a_j$. Then $h(pa_1 a_2 \dots a_{j-1} a_{j+1} \dots a_n) = h(v)$.

So in both cases, there exists an action p in the persistent set in $cont(w', L)$ and a partial computation u that is one action shorter than v such that $h(pu) = h(v)$. Because p is in the persistent set in $cont(w', L)$, $w'p$ is in R_L. By induction, there is $u' \in cont(w'p, R_L)$ such that $h(u') = h(u)$. Hence pu' is in $cont(w', R_L)$ and $h(pu') = h(pu) = h(v)$. □

Lemma 6. *For all v in $cont(w', L)$ there exists v' in $cont(w', R_L)$ such that $h(v) = h(v')$ and $h(cont(w'v, L)) = h(cont(w'v', R_L))$.*

Proof. We proof this lemma by induction on the length of $v \in cont(w', L)$. The basis is immediate since ε is in $cont(w', L)$ as well as in $cont(w', R_L)$ and, by Lemma 5, $h(cont(w', L)) = h(cont(w', R_L))$.

If $|v| > 0$ then $v = a_1 a_2 a_3 \dots a_n$ for actions $a_i \in \Sigma$. Assume that all a_i are not in the persistent set in $cont(w', L)$. Let $p \in \Sigma$ be in the persistent set in $cont(w', L)$. Then $vp \in cont(w', L)$. Action p is independent to all a_i and therefore pv is also in $cont(w', L)$ as well as is $pa_1 a_2 \dots a_{n-1}$. Because p can only be independent of a_i if $h(p) = h(a_i)$, $h(vp) = h(pv)$ which implies that $h(pa_1 a_2 \dots a_{n-1}) = h(v)$.

If some a_i are in the persistent set in $cont(w', L)$ then let j be the position of the first action in v that is in the persistent set $cont(w', L)$. Let $p = a_j$. Then $h(pa_1 a_2 \dots a_{j-1} a_{j+1} \dots a_n) = h(v)$.

So in both cases, there exists an action p in the persistent set in $cont(w', L)$ and a computation u that is one action shorter than v such that $h(pu) = h(v)$. Because p is in the persistent set in $cont(w', L)$, $w'p$ is in R_L. By induction, there is $u' \in cont(w'p, R_L)$ such that $h(u') = h(u)$ and $h(cont(w'pu, L)) \subseteq h(cont(w'pu', R_L))$. Hence pu' is in $cont(w', R_L)$ and $h(pu') = h(pu) = h(v)$ and $h(cont(w'pu, L)) \subseteq h(cont(w'pu', R_L))$. □

The lemmas above imply that a persistent-set reduction with respect to an h-compatible independence relation Δ results in an h-compatible partial-order reduction. So persistent-set reductions are a feasible approach to the construction of an abstract state-space. Then, from Theorem 2, it follows that the simplicity of an abstraction on a partial behaviour can be checked on a persistent-set reduction to it and thus we can check approximately satisfied properties of a behaviour on a simple abstraction of its persistent-set reduction:

Corollary 3. *If $h(R_L)$ does not contain maximal words then the condition*

$$lim(h(R_L)) \models_{RL} \mathcal{P} \quad \textit{if and only if} \quad lim(L) \models_{RL} h^{-1}(\mathcal{P})$$

holds if and only if h is simple on R_L.

Corollary 3 states that we can check properties of behaviour $lim(L)$, even under fairness, without computing a complete representation of it. Computing a representation for R_L is sufficient.

6 Conclusion

We have shown in this paper that the persistent-set technique [5,12] offers a practical approach to compute abstraction compatible reduced state spaces [10]. The reduced state space enables us to compute the same abstraction as for the complete state space, including checking the simplicity of the abstraction (simplicity guarantees the usefulness of an abstraction with respect to verification). Taking into account that persistent sets can be computed effectively, for instance by using stubborn-sets [11], we have found indeed a practical way for abstract state-space construction without an exhaustive construction of the concrete one, filling the gap left in [10]. It is very likely that, by a modification of the abstraction-compatible persistent-set reduction presented in this paper, more powerful reductions can be achieved in a practical way. However, this will be a topic of further study.

References

1. B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985. 471, 471, 471
2. E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. In *Proceedings of the 19th ACM Symposium on Principles of Programming Languages*, Albuquerque, 1992. 470
3. V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, first edition, 1995. 473
4. S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, New York, 1974. 471, 471
5. P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. *Formal Methods in System Design*, 2(2):149–164, April 1993. 470, 470, 475, 477
6. T. A. Henzinger. Sooner is safer than later. *Information Processing Letters*, 43:135–141, 1992. 471, 471
7. U. Nitsche and P. Ochsenschläger. Approximately satisfied properties of systems and simple language homomorphisms. *Information Processing Letters*, 60:201–206, 1996. 470, 471, 471, 471, 471, 472, 472, 473
8. U. Nitsche and P. Wolper. Relative liveness and behavior abstraction (extended abstract). In *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC'97)*, pages 45–52, Santa Barbara, CA, 1997. 470, 471, 471, 472, 472, 472

9. P. Ochsenschläger. Verification of cooperating systems by simple homomorphisms using the product net machine. In J. Desel, A. Oberweis, and W. Reisig, editors, *Workshop: Algorithmen und Werkzeuge für Petrinetze*, pages 48–53. Humboldt Universität Berlin, 1994. 472
10. U. Ultes-Nitsche. Towards the construction of an abstract state-space from a partial-order representation of the concrete one. *Electronic Notes in Theoretical Computer Science*, 18:1–17, 1998. 470, 470, 477, 477
11. A. Valmari. A stubborn attack on state explosion. In E. M. Clarke and R. P. Kurshan, editors, *CAV'90—Computer Aided Verification 1990*, volume 531 of *Lecture Notes in Computer Science*, pages 156–165. Springer Verlag, 1991. 470, 477
12. P. Wolper and P. Godefroid. Partial-order methods for temporal verification. In E. Best, editor, *CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 233–246. Springer Verlag, 1993. 470, 470, 475, 477

Computational Power of Neuroidal Nets*

Jiří Wiedermann

Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic
wieder@uivt.cas.cz

Abstract. Neuroid as a kind of a programmable neuron has been introduced by L. G. Valiant in 1988. Essentially it is a combination of a standard threshold element with a mechanism that allows for modification of neuroid's computational behaviour. This is done by changing the settings of its weights and of its threshold in the course of computation. It is shown that the computational power of neuroidal nets crucially depends on the size of allowable weights. For bounded weights their power equals to that of that of finite automata, whereas for unbounded weights finite neuroidal nets posses the computational power of Turing machines. It follows that the former neuroidal nets are computationally equivalent to standard, non-programmable discrete neural nets, while, quite surprisingly, the latter nets are computationally equivalent to a certain kind of analog neural nets.

1 Introduction

Nowadays, neural nets belong among computational models that find their use primarily in the field of so-called cognitive computing. Cognitive computing is the notion coined by L.G. Valiant [6] to denote any computation whose computational mechanism is based on our ideas about brain computational mechanisms and whose goal is to model cognitive abilities of living organisms. Numerous variants of neural nets have been proposed and studied. They differ in the computational properties of their basic building elements, viz. neurons. Usually, two basic kinds of neurons are distinguished: discrete ones, that compute with Boolean values 0 and 1, and analog (or continuous) ones that compute with any real or rational number between 0 and 1.

As far as the computational power of the respective neural nets is concerned, it has been known that finite nets consisting of discrete neurons are computationally equivalent to finite automata (cf. [3]). On the other hand, finite nets of analog neurons, computing in discrete steps with rational values, are computationally equivalent to Turing machines (cf. [2]). When computations with reals and continuous time is considered than the respective analog nets posses even super-Turing computational abilities [2]. No type of finite discrete neural nets has been known that would be more powerful than finite automata.

* This research was supported by GA ČR Grant No. 201/98/0717.

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 479–487, 1999.

An important aspect of all interesting cognitive computations is learning. Neural nets learn by adjusting the weights on neural interconnections according to a certain learning algorithm. This algorithm and the corresponding mechanism of weight adjustment has not been considered as part of the network.

Inspired by real biological neurons Valiant suggested in 1988 [4] a special kind of programmable discrete neurons, called neuroids, in order to make learning mechanism a part of neural nets. Based on its current state and current excitation from firings of neighboring neuroids, a neuroid can in the next step change all of its computational parameters (i.e., can change its state, threshold, and weights). In his monograph [5] Valiant suggested a number of neuroidal learning algorithms that demonstrate a viability of neuroids. Nevertheless, sufficient attention has not been paid to computational power of the respective nets. Without pursuing this idea any further Valiant merely mentioned that computational power of neuroids depend on the restriction put upon possibilities of neuroids to self-modify their computational parameters.

In this paper we shall study the computational power of neuroidal nets w.r.t to restrictions on update abilities of neuroidal computational parameters.

We shall prove that finite neuroidal nets with weights of size $S(n)$ that allow a simple arithmetic in their weight (i.e., doubling and integer halving of weights, and increasing of a weight by 1) are computationally equivalent to computations of any $S(n)$-space bounded Turing machine. Contrary to that, if we restrict the values of each neuroid's parameter to a fixed set, then the resulting neuroidal net (which is called by Valiant as a "simple complexity theoretic model") is computationally equivalent to a finite automaton.

A formal definition of neuroidal nets is given in Section 2. In Section 3 we sketch the proof of computational equivalency of "simple complexity theoretic model" with finite automata. In Section 4 we show a simulation of a space bounded Turing machine by a finite neuroidal net that allows simple arithmetic on its weights. Finally, in conclusions we shall discuss the merits of the presented results.

A more elaborated version of this paper, containing new results that have been obtained after submitting this paper, appears in [8]. Among others, the new results give the direct simulation of discrete neuroidal nets by neural ones and the proof of the super-Turing computing power of finite neuroidal nets with real weights.

2 Neuroidal Nets

In what follows we shall define neuroidal nets making use of original Valiant's proposal [5], inclusively his notation.

Definition 1. A neuroidal net $\mathcal{N}$ *is a quintuple* $\mathcal{N} = (G, W, X, \delta, \lambda)$, *where*

- $G = (V, E)$ *is the directed graph describing the topology of the network;* V *is a finite set of* N *nodes called* neuroids *labeled by distinct integers* $1, 2, \ldots, N$, *and E is a set of M directed edges between the nodes. The edge (i, j) for $i, j \in \{1, \ldots, N\}$ is an edge directed from node i to node j.*

- *W is the set of numbers that are called* weights*. To each edge $(i, j) \in E$ there is a value $w_{i,j} \in W$ assigned at each instant of time.*
- *X is the finite set of* modes *of neuroids that a neuroid can be in each instant. Each mode is specified as a pair (q, p) of values where q is a member of a set Q of states, and p is an integer from a finite set T called the set of* thresholds *of the neuroid.*

  Q consists of two kinds of states called firing *and* quiescent *states.*

  To each node i there is also a Boolean variable f_i to have value one or zero depending on whether the node i is in a firing state or not.
- *δ is the recursive* mode update function *of form $\delta \colon X \times W \to X$.*

  Let $w_i \in W$ be the sum of those weights w_{ki} of neuroid i that are on edges (k, i) that come from neuroids that are currently firing, i.e., formally

  $$w_i = \sum_{\substack{k \text{ firing} \\ (k,i) \in E}} w_{ki} = \sum_{\substack{j \\ (j,i) \in E}} f_j w_{(j,i)}$$

  The value of w_i is called the excitation *of i at that time.*

  The mode update function δ defines for each combination (s_i, w_i) that holds at time t, the mode $s' \in X$ that neuroid i will transit to at time $t + 1$: $\delta(s_i, w_i) = s'$.
- *λ is the recursive* weight update function *of form $\lambda \colon X \times W \times W \times \{0, 1\} \to W$. It defines for each weight w_{ji} at time t the weight w'_{ji} to which it will transit at time $t + 1$, where the new weight may depend on the values of each of s_i, w_i, w_{ji}, and f_j at time t: $\lambda(s_i, w_i, w_{ji}, f_j) = w'_{ji}$*

The elements of sets Q, T, W, and f_i's are called *parameters* of the net $\mathcal{N}$.

A *configuration* of $\mathcal{N}$ at time t is the list of modes of all neurons followed by the list of weights of all edges in $\mathcal{N}$ at that time. The respective lists of parameters are pertinent to neuroids ordered by their labels and to edges ordered lexicographically w.r.t. the pair of labels (of neuroids) that identify the edge at hand. Thus, at any time a configuration is an element from $X^N \times W^M$.

The *computation of a neuroidal network* is determined by the *initial conditions* and the *input sequence*. The initial conditions specify the initial values of weights and modes of the neuroids. These are represented by initial configuration. The input sequence is an infinite sequence of inputs or *stimuli* that specifies for each $t = 0, 1, 2, \ldots$ a set of neuroids along with the states into which these neuroids are forced to enter (and hence forced to fire or prevented from firing) at that time by mechanisms outside the net (by peripherals).

Formally, each stimulus is an N-tuple from the set $\{Q \cup *\}^N$. If in the t-th N-tuple s_t there is a symbol q at i-th position then this denotes the fact that the neuroid i is forced to enter state q at time t. The special symbol $*$ is used as don't care symbol at positions which are not influenced by peripherals at that time.

A *computational step* of a neuroidal net $\mathcal{N}$ that finds itself in a configuration c_t and receives its input s_t at time t is performed as follows. First, neuroids are forced to enter into states as dictated by the current stimuli. Neurons not influenced by peripherals at that time retain their original state as in the configuration c_t. In this way a new configuration c'_t is obtained. Now for this configuration the excitation w_i is computed and mode and weight updates are realized for each neuroid i in parallel, in accordance with the respective function δ and λ. In this way a new configuration c_{t+1} is entered.

The result of the computation after the t-th step is the N-tuple of states of all neuroids in c_{t+1}. This N-tuple is called the *action* at time t. Obviously, any action is an element in Q^N. Then the next computational step can begin.

The output of the whole computation may be seen as an infinite sequence of actions.

From computational point of view any neuroidal net can be seen as a transducer that reads an infinite sequence of inputs (stimuli) and produces an infinite sequence of outputs (actions).

For more details about the model see [5].

3 Variants of Neuroidal Nets

In the previous definition of neuroidal nets we allowed the set W to be any set of numbers and weight and mode update functions to be arbitrary recursive functions. Intuitively it is clear that by restricting these condition we will get variants of neural nets that will differ in their expressiveness as well as in their computing power. In his monograph Valiant [5] discusses this problem and essentially suggests two extreme possibilities.

The first one considers the neuroidal nets where the set of weights of individual neuroids is finite. This is called a "simple complexity-theoretic model" in Valiant's terminology. We shall also call the respective model of a neuroid as a "bounded weight" neuroid. Note that in this case the function δ and λ can both be described by finite tables.

In some sense an opposite possibility is to consider the set W as the set of all integers. In this case it is no longer possible to describe the weight update function by a finite table. Rather, what we need is a suitable recursive function that will allow for efficient weight modifications. Therefore, we will consider a weight update function that allows for weight doubling (i.e., multiplication by 2), integer halving (integer division by 2), and adding or subtracting 1. Such a weight update function will be called a *simple-arithmetic update function.* The respective neuroid will be called as "unbounded weight" neuroid. The *size* of each weight will be given by the number of bits needed to specify the respective weight value.

4 Bounded Weight Neuroidal Nets and Finite Automata

It is obvious that in the case of neuroidal nets with bounded weights there is but a final number of different configurations a single neuroid can enter. Hence, its computational activities, similarly as those of a any finite neuroidal net, can be described by a single finite automaton (or more precisely: by a finite transducer). In order to get some insight into the relation between the sizes of the respective devices we shall describe the construction of the respective transducer in more detail in the next theorem. In fact this transducer will be a Moore machine (i.e., the type of a finite automaton that produces an output after each transition) since there is an output (action) produced by $\mathcal{N}$ after each computational move.

Theorem 1. *Let $\mathcal{N}$ be a finite neuroidal net consisting of N neuroids and M edges of a bounded weight. Then there is a constant $c > 0$ and a finite Moore automaton $\mathcal{A}$ of size $O(c^{N+M})$ that simulates $\mathcal{N}$.*

Proof (sketch). We shall describe the construction of the Moore automaton $\mathcal{A} = (I, S, q_0, O, \Delta)$. Here I denotes the input alphabet whose elements are N-tuples of stimuli: $I = \{Q \cup *\}^N$. The set S is the set of states consisting of all configurations of $\mathcal{N}$, i.e., $S = X^N \times W^M$. The state q_0 is the initial state and it is equal to the initial configuration of $\mathcal{N}$. The set O denotes the set of outputs of $\mathcal{A}$. It will consist of all possible actions of $\mathcal{N} : O = Q^N$.

The transition function $\Delta \colon I \times S \to S \times O$ is defined as follows: $\Delta(i, s_1) = (s_2, o)$ if and only if the neuroid $\mathcal{N}$ in configuration s_1 and with input i will enter the configuration s_2 and produce the output o in one computational move. It is clear that the input-output behaviour of both $\mathcal{N}$ and $\mathcal{A}$ is equivalent. □

Note that the size of the automaton is exponential w.r.t size of the neuroidal net. In some cases such a size explosion seems to be unavoidable. For instance, a neuroidal net consisting of N neuroids can implement a binary counter that can count upto c^N, where $c \geq 2$ is a constant that depends on the number of states of the respective neuroids. The equivalent finite automaton would then require at least $\Omega(c^N)$ states. Thus, the advantage of using neuroidal nets instead of finite automata seems to lay in the description economy of the former devices. A similar observation has been made for standard neural nets in [3]. Since finite standard neural nets recognize exactly regular languages (cf. [3]), we get the following consequence:

Corollary 1. *The computational power of neuroidal nets of finite type is equivalent to that of standard non-programmable neural nets.*

Thus, when compared to standard nets, it appears that programmability of neuroids of finite types does not increase their computational power; it contributes merely to their greater descriptive expressiveness.

5 Neuroidal Nets with Finite Weights and Turing Machines

Next we shall show that in the case of unbounded weights there exist neuroidal nets of finite size that can simulate any Turing machine.

Since we shall be interested in space-bounded machines w.l.o.g. we shall first consider a single-tape Turing machine in place of a simulated machine. In order to extend our results also for sublinear space complexities we shall later consider also single-tape machines with a separate input tape.

First we show that even a single neuroid is enough for simulation of a single tape Turing machine.

Theorem 2. *Any $S(n)$-space bounded single tape Turing machine*[1] *can be simulated in linear time by a single neuroid that makes use of integer weights of size $O(S(n))$ and of a simple arithmetic weight update function.*

Proof (sketch). Since we are dealing with space-bounded Turing machines (TMs), w.l.o.g. we can consider only single-tape machines. Thus, in what follows we shall describe simulation of one computational step of a single-tape Turing machine $\mathcal{M}$ of space complexity $S(n)$ with tape alphabet $\{0, 1\}$. It is known (cf. [1]) that the tape of such a machine can be replaced by two stacks, S_L and S_R, respectively. The first stack holds the contents of $\mathcal{M}$'s tape to the left from the current head position while the second stack represents the rest of the tape. The left and right end of the tape, respectively, find itself at the bottom of the respective stacks. Thus, we assume that $\mathcal{M}$'s head always scans the top of the right stack. For technical reasons we shall add an extra symbol 1 to the bottom of each stack. During its computation $\mathcal{M}$ merely updates the top, or pushes or pops the symbols to or from the top of these stacks.

With the help of a neuroid n we shall represent the machine $\mathcal{M}$ in a configuration that is described by the contents of its two stacks and state of the machine's finite state control in the following way. The contents of both stacks will be represented by two integers v_L and v_R, respectively. Note that both $v_L, v_R \geq 1$ thanks to 1's at the bottoms of the respective stacks. The instantaneous state of $\mathcal{M}$ is stored in states of n.

The respective neuroid n has four inputs i_1, i_2, i_3, and i_4. Referring to the previous notation, these inputs are connected to n via four connections, carrying weights $w_1 = v_L$, $w_2 = -v_L$, $w_3 = v_R$ and $w_4 = -v_R$, respectively. The output of n is connected to all four inputs.

The neuroid simulates each move of $\mathcal{M}$ in a series of a few moves. At the beginning of a series that will simulate the $(t+1)$-st move of $\mathcal{M}$, the following invariant is preserved by n, for any $t > 0$. The weights w_1 and w_3 represent the contents of stacks after the t-th move and $w_2 = -w_1$ and $w_4 = -w_3$. Initially, at the beginning of computation, the left stack is empty and the right stack

[1] Note that in case of single tape machines the input size is counted into the space complexity and therefore we have $S(n)) \geq n$.

contains the input word of $\mathcal{M}$. We shall assume that n will accept its input by entering a designated state

Assume that at time t the finite control of $\mathcal{M}$ is in a state q. Until its change this state is stored in all forthcoming neuroidal states that n will enter.

In order to read the symbol from the top of the right stack the neuroid has to determine the last binary digit of w_3 or, in other words, it has to determine the parity of w_3. This is done by the following algorithm (in the sequel, the threshold of n is permanently set to 0.)

- *Prepare for comparison and fire:* In a parallel step, n performs the following updates: $w_3 := (w_3 + 1) \div 2$ and $w_4 := w_4 \div 2$ and fires;
- *Compare:* The total excitation, i.e., the sum w of all four weights is computed and compared again the threshold. Clearly, $w_1 + w_2 = 0$ and therefore $w > 0$ iff w_3 was odd at time t;
- *Rewrite:* For definiteness assume that $w > 0$ in the previous step and let the transition to be made by $\mathcal{M}$ in this move be $\delta(q, 1) = (s, 0, L)$, say. This information is stored in n's finite control. Based on this, the neuroid will perform the following action. In order to rewrite the top of the right stack accordingly, w_3 has to be doubled, and w_4 has to be updated correspondingly as well. This is achieved by updating the weights $w_3 := 2w_3$ and $w_4 := 2w_4$. If the move was $\delta(q, 1) = (s, 1, L)$, then instead of the previous updates the updates $w_3 := 2w_3 + 1$ and $w_4 := 2w_4 + 1$ should have been performed.
- *Move:* Move to the left (right) is performed by first reading and deleting the top symbol from the left (right) stack and its pushing onto the right (left) stack. This is realized by changing all the weights with the help of appropriate updates similarly as we did before.
- *Accept or simulate $\mathcal{M}$'s transition into the next state:* If s is accepting then halt and accept, otherwise store s into the state of neuroid's finite control and simulate the next, $t + 1$-st move of $\mathcal{M}$.

  It is clear that the simulation runs in linear time w.r.t. the original Turing machine time complexity and that size of any stack never exceeds the value $S(n) + 1$. Hence the size of the weights of n will be bounded by the same value.

□

Note that the previous idea of simulation would work even in the case when only the operation of weight increase or decrease by 1 would have been at our disposal. In this case the contents of the tape (or stacks) would be represented in unary notation. This would lead to exponential weights sizes and exponential simulation time.

Also note that the similar construction, with still using only one neuroid, would work also in the case when a multiple tape Turing machine had to be simulated. To simulate a k-tape machine the resulting neuroid will represent each tape by 4 weights similarly as before. This will lead to a neuroid with $4k$ incoming edges.

Next we shall show that a simulation of an off-line Turing machine by a finite neuroidal network with unbounded weights is possible. This will enable to prove a similar theorem as before that holds for arbitrary space complexities:

Theorem 3. *Let $\mathcal{M}$ be an off-line multiple tape Turing machine of space complexity $S(n) \geq 0$. Then $\mathcal{M}$ can be simulated in linear time by a finite neuroidal net that makes use of integer weights of size $O(S(n))$ and of a simple arithmetic weight update function.*

Proof (sketch). In order to read the respective inputs the neuroidal net will be equipped with the same input tape as the simulated Turing machine. The simulating net will contain, except the neuroid n that takes care of proper update of stacks that represent the respective machine tapes, also two extra neuroids that implement the control mechanism of input head movement.

For each move direction (left or right) there will be a special – so-called move neuroid – that will fire if and only if the input head has to move in the respective direction. The symbol read by input head will represent an additional input to the neuroid n that simulates the moves of $\mathcal{M}$.

The information about the move direction will be inferred by the neuroid n. As seen from the description of the simulation in the previous theorem, n keeps track on that particular transition of $\mathcal{M}$ that should be realized during each series of it steps that simulates one move of $\mathcal{M}$.

Since s can transmit this information to the respective move neuroids only via firing and cannot distinguish between the two target neuroids, we will have to realize a (finite) counter in each move neuroid. The counter will count the number of firings of s that occur in an uninterrupted sequence. Thus, at the end of the last step of $\mathcal{M}$'s move simulation (see the proof of the previous theorem) s will send two successive firings to denote the left move and three firings for the right move. The respective signals will reach both move neuroids, but with the help of counting they will distinguish which of them is in charge for moving the head. Some care about synchronization of all three neuroids must be taken. □

It is clear that the computations of finite neuroidal nets with rational weights can be simulated by Turing machines. Therefore the computational power of both devices is the same.

In 1995, Siegelmann and Sonntag proved that the computational power of certain analog neural nets is equivalent to that of Turing machines. They considered finite neural nets with fixed rational weights. At time t, the output of their analog neuron i is a value between 0 and 1 that is determined by applying a so-called *piecewise linear activation* function ψ to the excitation w_i of i at that time (see the definition 1): $\psi\colon w_i \to \langle 0, 1\rangle$. For negative excitation, ψ takes the value 0, for excitation greater than 1 the value 1, and for excitations between 0 and 1, $\psi(w_i) = w_i$. The respective net computes synchronously, in discrete time steps.

We shall call the respective nets as synchronous analog neural nets.

Siegelmann and Sonntag's analog neural network simulating a universal Turing machine consisted of 883 neurons. This is to be compared with the simple construction from Theorem 2 requiring but a single neuroid. Nevertheless, the equivalency if both type of networks with Turing machines proves the following corollary:

Corollary 2. *Finite synchronous analog neural nets are computationally equivalent to finite neuroidal nets with unbounded weights.*

6 Conclusions

The paper brings a relatively surprising result showing computational equivalence between certain kinds of discrete programmable and analog finite neural nets. This result offers new insights into the nature of computations of neural nets.

First, it points to the fact that the ability of changing weights is not a condition *sine qua non* for learning. A similar effect can be achieved by making us of reasonably restricted kinds of analog neural nets.

Second, the result on the computational equivalency of the respective nets supports the idea that all reasonable computational models of the brain are equivalent (cf. [7]).

Third, due to their transparency, for modeling of cognitive or learning phenomena, the programmable neuroidal nets seem to be preferred. The fixed weight nets seem to be more appropriate for neuromorphic implementation.

The direct simulation between neuroidal nets and analog neural nets presents an interesting open problem.

References

1. Hopcroft, J. E. – Ullman, J. D.: Formal Languages and Their Relation to Automata. Addison-Wesley, Reading, Mass., 1969. 484
2. Siegelmann, H. T. – Sonntag, E. D.: On Computational Power of Neural Networks. *J. Comput. Syst. Sci.*, Vol. 50, No. 1, 1995, pp. 132–150. 479, 479
3. Šíma, J. – Wiedermann, J.: Theory of Neuromata. Journal of the ACM, Vol. 45, No. 1, 1998, pp. 155–178. 479, 483, 483
4. Valiant, L.: Functionality in Neural Nets. Proc. 7th Nat. Conf. on Art. Intelligence, AAAI, Morgan Kaufmann, San Mateo, CA, 1988, pp. 629–634. 480
5. Valiant, L. G.: Circuits of the Mind. Oxford University Press, New York, Oxford, 1994, 237 p., ISBN 0–19–508936–X. 480, 480, 482, 482
6. Valiant, L. G.: Cognitive Computation (Extended Abstract). In: Proc. 38th IEEE Symp. on Fond. of Comp. Sci., IEEE Press, 1995, p. 2–3. 479
7. Wiedermann, J.: Simulated Cognition: A Gauntlet Thrown to Computer Science. To appear in ACM Computing Surveys, 1999. 487
8. Wiedermann, J.: The Computational Limits to the Cognitive Power of Neuroidal Tabula Rasa. To appear in: Proc. of the 10th Int. Conf. on Algorithmic Learning Theory ALT '99, Springer-Verlag, Berlin, 1999. 480

Cellular Automata with Dynamically Reconfigurable Buses

Thomas Worsch

Universität Karlsruhe, Fakultät für Informatik
worsch@ira.uka.de

Abstract. We consider one-dimensional cellular automata which are extended by dynamically reconfigurable buses (RCA). It is shown that up to a constant factor it does not matter whether bus segments are directed or not. For both variants of the model their time complexity can be characterized in terms of the reversal complexity of one-tape TM. The comparison of RCA with tree CA shows that the former are in the second machine class and that they can be transformed in some normal form with an only polynomial overhead of time.

1 Introduction

Dynamically reconfigurable architectures have received growing attention during the last years. A lot of different models have been investigated. They all have in common that some kind of processors are exchanging information on global buses the structure of which can be modified in each step by each processor by segmenting and/or fusing bus segments locally.

One of the first models was introduced by Rothstein [9] under the name of *bus automata*. In a one-dimensional array of finite automata (cells) neighboring cells are connected by a finite number of bus segments. It is the characteristic of a bus that information travels over arbitrary distances in constant time.

Most of the models considered in the sequel differ in two respects. The processors are RAM with a memory of non-constant size (i.e. depending on the input size). And the models are two- or even higher-dimensional. Sometimes the array of processors itself is d-dimensional, as e.g. for polymorphic processor arrays [5] or reconfigurable meshes [1,6], or the processors are extending in one dimension and a non-constant number of buses in the second as in the reconfigurable multiple bus machine [11] and the distributed memory bus computer [10]. Uniform families of finite reconfigurable networks have been investigated by Ben-Asher et al. [2]. In some cases it can be shown that these systems can (in some sense) simulate PRAM in linear time and vice versa, provided that both models use corresponding modes (exclusive vs. concurrent) for reading from and writing to buses resp. global memory (see e.g. [1,11]).

In this paper we are interested in the case of one-dimensional systems of finite automata with reconfigurable buses which we call *reconfigurable cellular automata* for short. It shown that even such systems (without a large bisection

J. Pavelka, G. Tel, M. Bartošek (Eds.): SOFSEM'99, LNCS 1725, pp. 488–496, 1999.

width, i.e. bandwidth) are in the second machine class. In addition observations made by Rothstein [9] concerning the relation of RCA to TM are generalized.

The rest of the paper is organized as follows. In Sect. 2 reconfigurable cellular automata are introduced. In Sect. 3 polynomial time simulations between RCA and tree CA (TCA) are established and their consequences are discussed. In Sect. 4 the relation of (both variants of) RCA to sequential TM is investigated.

2 Basic Notions

A *reconfigurable cellular automaton* with k bus segments (k−RCA) consists of a one-dimensional array of finite automata (cells) working synchronously according to the same local rule. Q denotes the set of states of one cell and B the bus alphabet. Cells are connected by bus segments. In the case of bidirectional bus segments there are k such segments between each two immediately neighboring cells and we write R$_2$CA. In the case of unidirectional bus segments there are k bus segments for information transmission from cell i to $i+1$ and k segments in the opposite direction. The model is denoted as R$_1$CA.

One step of each cell consists of four phases:

1. **Configure:** Depending on the local state, arbitrary subsets of (locally available) bus segments are connected.
2. **Send:** Depending on the local state, on each such subset the cell may send a symbol $b \in B$ or send nothing.
3. **Receive:** On each bus segments the cell may observe a symbol sent by a cell participating in the bus.
4. **New state:** Depending on the old state and the symbols received (if any) the cell enters a new state.

In the case of R$_2$CA a symbol sent spreads along all bus segments belonging to the bus on which it was sent. In an R$_1$CA a symbol only spreads along the directions of the segments. In both cases it may happen that cells want to send different symbols on the same bus. One can distinguish different variants depending on the conflict resolution strategy employed. As a matter of fact in all the constructions described below, one can always guarantee that there is always at most one cell sending a symbol on a bus. No conflicts ever happen.

Moshell and Rothstein [7] consider a more general version of bus automata where each cell participating in the communication on a bus can execute certain computations modifying the symbol received by cells. We do not allow this.

Remark 1. Given an R$_2$CA one can construct an R$_1$CA simulating it in real time by simply replacing each bidirectional bus segment by two unidirectional bus segments going in opposite directions.

3 RCA Are in the Second Machine Class

In this section R$_2$CA will be shown to belong to the second machine class, i.e. the problems solvable by them in polynomial time are exactly those in PSPACE. This will be done by describing polynomial time simulations of and by tree-shaped CA (TCA).

That trees as an underlying topology of one kind or the other are a powerful tool has been shown by several authors. Wiedermann [12] has shown that parallel TM with tree-shaped tapes are in the second machine class. For a model consisting of a full ternary tree of finite automata receiving the input in parallel along one branch this has been shown by Mycielski and Niwiński [8]. For binary trees with sequential input at the root node, Fellah and Yu [4] describe a linear time correspondence with ATM. Therefore for these models $\mathrm{TCA{-}TIME}(\mathrm{Pol}(n)) = \mathrm{PSPACE}$ is known.

Theorem 1. *Let t be a function satisfying the requirements of Lemmata 1 and 2. Then:*

$$\mathrm{R_2CA{-}TIME}(\mathrm{Pol}(t)) = \mathrm{TCA{-}TIME}(\mathrm{Pol}(t)) \ .$$

Remark 2. It immediately follows that $\mathrm{RCA{-}TIME}(\mathrm{Pol}(n)) = \mathrm{PSPACE}$, i.e. $\mathrm{R_2CA}$ are in the second machine class.

For the proof of the theorem we use the following two lemmata.

Lemma 1. *Let t be a function such that a* TCA *can mark the cells in a subtree of height* $\mathrm{O}(t)$. *Then:*

$$\mathrm{R_2CA{-}TIME}(t) \subseteq \mathrm{TCA{-}TIME}(\mathrm{O}(t^3)) \ .$$

Sketch of proof. The cells of an $\mathrm{R_2CA}$ R are mapped to cells of a TCA T as indicated in Fig. 1. The simulation of one step of R is done in a number of steps proportional to the square of the logarithm of the maximum number of cells used by R, and hence needs time $\mathrm{O}(t^2)$.

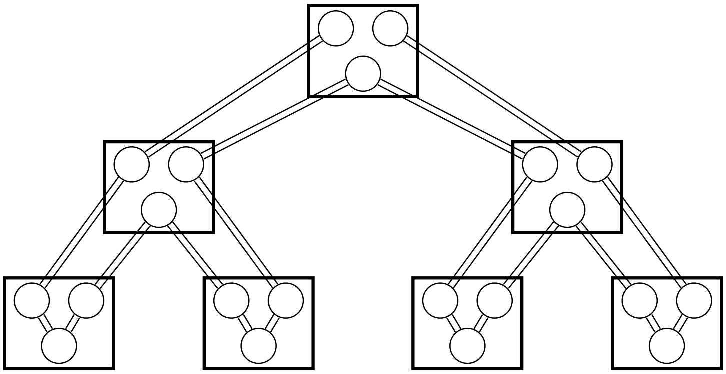


Fig. 1. Mapping cells of an $\mathrm{R_2CA}$ (circles) to cells of a TCA (rectangles)

First for each T-cell i a relation P_i on all bus segments belonging to the corresponding R-cells is computed such that $(s, s') \in P_i$ iff a symbol sent on s is transmitted to s' (assuming that no symbol would be sent on any other segment of the bus connecting s and s'). These can be computed by sending for $1 \leq j \leq t$ a wave front W_j starting at the leaves, moving up i levels and back to the leaves. W_j ensures that $(s, s') \in P_i$ if two segments (s, s') in one T-cell are connected by a path along a bus which passes only through R-cells stored in T-cells somewhere in the j lowest levels of T.

In each T-cell P_i is initialized with those (s, s') for which an R-cell locally connects them. Then W_j updates P_i by constructing the transitive closure of $P_i \cup P'$, where P' is the union of the restrictions of all $P_{i'}$ of previously visited T-cells to those segments which connect them with T-cell i. Once the relations are computed the simulation of sending symbols along the buses can be simulated in time $\mathrm{O}(t)$. □

Lemma 2. *Let t be a function such that an* $\mathrm{R_2CA}$ *can mark 2^t cells in time* $\mathrm{O}(t^2)$. *Then:*

$$\mathrm{TCA{-}TIME}(t) \subseteq \mathrm{R_2CA{-}TIME}(\mathrm{O}(t^2)) \ .$$

Sketch of proof. The cells of a TCA T are mapped to cells of an $\mathrm{R_2CA}$ R as indicated in Fig. 2. The simulation of one step of T is done in a number of phases. Each phase takes a constant number of steps and is used to simulate the exchange of data between two successive layers of cells of T. Thus the overall time for the simulation of one step of T is proportional to the height of T.

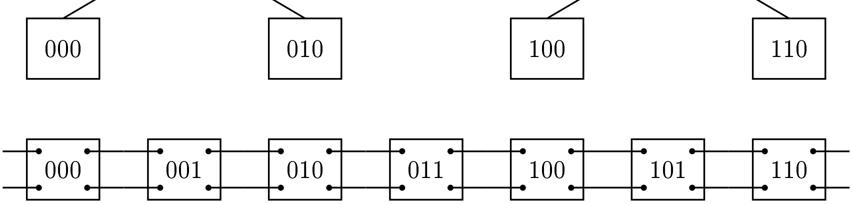


Fig. 2. Mapping TCA cells to $\mathrm{R_2CA}$ cells for a tree of height 2

In the first phase the exchange of data between the leaves and their parent nodes is simulated. For the following description cells are assumed to be numbered starting at 0 at the left end. Nodes with numbers $4i$, $i \in \mathbf{N}$ are marked as left leaves, nodes $4i+2$ as right leaves, nodes $4i+1$ as parent nodes and nodes $4i+3$ as not relevant (see Algorithm 2 below). No bus segments are connected.

In the next phase cells which previously represented leaves connect their left with the corresponding right bus segments and don't participate in the rest of the simulation of this step of T. Analogously as described for the first phase the remaining cells are marked and simulate the exchange of data on the next level. Once all levels have been provided with the informations from other levels, all cells can enter their new states according to the local rule of T.

Details of the construction which are concerned with technicalities as for example handling of inputs and marking of certain segments of cells of R are omitted. Since the height of the used part of the tree can be bounded by the number of steps of T, the total running time of R is at most $\mathrm{O}(t^2)$. □

It remains to show how one can in constant time (depending only on m but not on the number of cells involved) designate for example every 2^mth cell. The following algorithm applies an extension of a well-known trick to compute the exclusive or of a number of bits on a reconfigurable array.

Algorithm 2. Consider a $2{-}\mathrm{R}_2\mathrm{CA}$ which starts in a configuration where each cell connects its left segment l_1 (resp. l_2) with the right segment r_2 (resp. r_1) as indicated in the top row of Fig. 3 below.

The cell at the left end sends a signal on l_1. This is received alternately on segment l_1 (resp. l_2) in even (resp. odd) numbered cells (if counting starts with 0 at the left end), and this is interpreted as a 0 (resp. 1). In subsequent steps cells with the connections $l_1{-}r_2$ and $l_2{-}r_1$ receiving a 0 change them to $l_1{-}r_1$ and $l_2{-}r_2$. Other cells don't change their bus configuration.

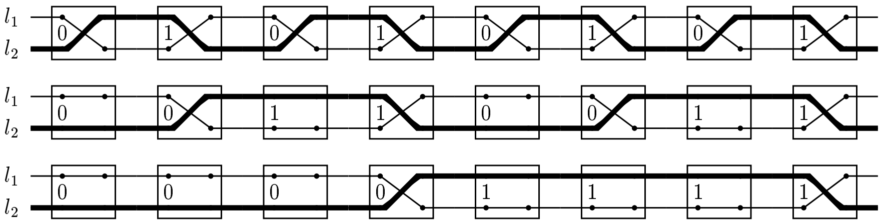


Fig. 3. Generating regular patterns of alternating blocks of 0 and 1

It can be easily seen that the resulting patterns of 0 and 1 are exactly those needed in the construction of Lemma 2. One can also observe that the sequence of bits "received" by a cell constitute the bits (from least significant to most significant) of the number of that cell.

Corollary 1. *Given an arbitrary* $\mathrm{R_2CA}$ *R one can use Lemmata 1 and 2 to construct a* $\mathrm{R_2CA}$ *R' (via a* TCA*) which simulates R in polynomial time and has certain "nice" properties:*

1. *In R' there are only* 2 *bus segments between neighbors (instead of k).*
2. *All buses of R' are linear (while buses of R may fork or form cycles).*
3. *In R' only exclusive sends happen (while in R common sends or even conflicts may occur).*

4 A Characterization of RCA Time by TM Reversal

Below we write $\mathrm{T_1TM}$ to denote TM with one work tape, one read-write head on it, and without a separate input tape.

Theorem 3. *If f satisfies the requirements of Lemma 3, then:*

$$\mathrm{R_2CA{-}TIME}(\Theta(f)) = \mathrm{R_1CA{-}TIME}(\Theta(f)) = \mathrm{T_1TM{-}REV}(\Theta(f)) \ .$$

Remark 3. Before we are going to prove this result, a few comments seem to be in order. First, one is provided with a characterization of RCA−TIME up to a constant factor and not only within polynomial bounds.

Second, the theorem states that one can not only simulate an $\mathrm{R_2CA}$ by an $\mathrm{R_1CA}$ in linear time as mentioned in Remark 1. The opposite direction is also possible which is much less obvious. This means that algorithms can be used equally well for either type of system. For example a VLSI realization may provide bidirectional bus segments while in an optical computer directed information transmission may be easier to implement.

Third, one should observe that (somewhat surprisingly) the characterization is in terms of reversal complexity of TM with only one work tape and without an input tape. Usually this model is considered to be too weak and multi-tape machines are used in connection with reversal complexity [3].

A proof of Theorem 3 can be given by establishing 3 set inclusion in a circular way "from left to right". The first part has been mentioned in Remark 1. The two other cases are treated in Lemmata 3 and 4 below.

Lemma 3. *Let f be a function such that a* $\mathrm{T_1TM}$ *can mark 2^t tape squares using* $\mathrm{O}(t)$ *reversals. Then an f time-bounded* $\mathrm{R_1CA}$ *can be simulated by an* $\mathrm{O}(f)$ *reversal-bounded* $\mathrm{T_1TM}$.

Sketch of proof. One has to show how to simulate one step of an $\mathrm{R_1CA}$ R with a constant number of reversals of a TM T. Cell states are stored on the tape. Below we sketch the main task, which is to construct in cell i a relation P_i similar to the one used in the proof of Lemma 1, but in the present case of $\mathrm{R_1CA}$ it is not necessarily symmetric and it only concerns the bus segments of one cell. Other details are omitted.

In each cell P_i is initialized with those (s, s') for which the cell locally connects them. Then T does 3 full sweeps over all cells, updating P_i by constructing the transitive closure of $P_i \cup P'_{i'}$, where $i' = i \pm 1$ is the previously visited cell and $P'_{i'}$ is the restriction of $P_{i'}$ to those segments which connect i' and i.

To see that a constant number of sweeps suffices to compute all final P_i consider any pair (s, s') and a shortest (partial) bus G leading from s to s'. G consists of linear parts $G_1, G_2, \ldots, G_m$ alternatingly leading from left to right and vice versa. After 2 sweeps all cells belonging e.g. to G_1 and G_2 know this fact, and analogously for other corresponding pairs (G_i, G_{i+1}). Therefore during the third sweep any still "missing" connections can be derived via the transitive closure. □

Lemma 4. *An f reversal-bounded* $\mathrm{T_1TM}$ *can be simulated by an* $\mathrm{O}(f)$ *time-bounded* $\mathrm{R_2CA}$.

Sketch of proof. Let T be a $\mathrm{T_1TM}$. Without loss of generality one may assume that T moves its head in every step. We show how a whole sweep of T between two head reversals can be simulated in constant time by a $\mathrm{R_2CA}$ R.

If only an $\mathrm{R_1CA}$ were required one could use Rothstein's idea [9]: Consider e.g. a sweep from left to right. The symbols of successive tape squares are stored in successive cells of R. For each state q of T there is a bus segment l_q entering from the left neighboring cell and a bus segment r_q leaving to the right. For all q and b, if T upon entering a square with symbol b in state q from the left will leave it to the right in state q' after having changed the symbol to b', then a cell storing b connects l_q with $r_{q'}$.

Then the cell responsible for the tape square on which the last reversal happened sends a signal on the segment r_q corresponding to the state on which T leaves the square. Since the bus segments are unidirectional the signal spreads along exactly one path from left to right, "activating" in each cell exactly that l-segment belonging to the state T is in when entering the corresponding tape square. Depending on the state and its stored symbol each cell can enter a new state representing the new tape symbol.

The direct transfer of this idea to $\mathrm{R_2CA}$ results in a failure. Since bus segments are not directed any longer, in each cell which connects at least two l_{q_1} and l_{q_2} to the same $r_{q'}$ the signal arriving on one of the l-segments will spread "backwards" along the others. As a result it may happen that some cells observe the signal on every bus segments. Fig. 4 shows an example of such a signal tree.

The black path is where the signal would only be seen, if the bus segments were directed. Due to the "undirectedness" of the bus segments the gray paths also carry the signal. These can be eliminated in constant time as follows. In each of several phases gray paths leading from a leaf to the first cell where another gray or black path leads to the left are cut off. These paths can be identified by the cells at the leaves because those cells know that they did not send a signal on it. Since the structure of the signal tree is restricted by the bounded number of bus segments between cells, a constant number of phases is sufficient to cut off all gray parts leaving only the desired black path. □

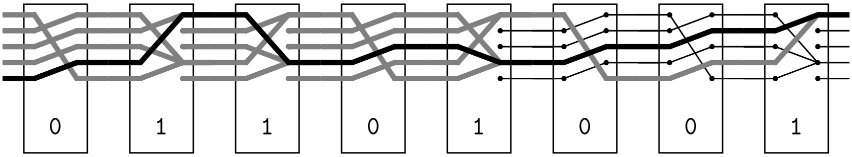


Fig. 4. First step of the simulation of a TM sweep in constant time on an R_2CA

5 Summary

It has been shown that one-dimensional RCA are in the second machine class. Their time complexity can be characterized in terms of reversal complexity of one-tape Turing machines. Via simulations by and of TCA one can obtain a normal form of RCA with $k = 2$, only linear buses and exclusive sending on the buses working with a polynomial overhead of time. Although the proof techniques cannot be carried over to the one-dimensional case, self-simulation results for two-dimensional reconfigurable networks indicate that at least the reduction of k to 2 might be possible even in linear time. This is one of the problems currently under investigation. Others include restrictions on the maximum allowed bus length or the assumption of a non-constant lower bound for the transmission of information on long buses in order to make the model more realistic.

References

1. Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster. The power of reconfiguration. *Journal of Parallel and Distributed Computing*, 13(2):139–153, 1991. 488, 488
2. Y. Ben-Asher, K.-J. Lange, D. Peleg, and A. Schuster. The complexity of reconfiguring network models. *Information and Computation*, 121:41–58, 1995. 488
3. J. Chen. The difference between one tape and two tapes: with respect to reversal complexity. *Theoretical Computer Science*, 73:265–278, 1990. 493
4. A. Fellah and S. Yu. Iterative tree automata, alternating Turing machines, and uniform boolean circuits: Relationships and characterizations. In H. Berghel et al., eds., *Proc. Symposium on Applied Computing. Volume 2*, p. 1159–1166, 1992. 490
5. M. Maresca. Polymorphic Processor Arrays. *IEEE Transactions on Parallel and Distributed Systems*, 4:490–506, 1993. 488
6. R. Miller, V. K. Prasanna-Kumar, D. Reisis, Q. F. Stout. Meshes with reconfigurable buses. In *Proc. Conf. on Advanced Research in VLSI*, p. 163–178, 1988. 488
7. J. M. Moshell and J. Rothstein. Bus automata and immediate languages. *Information and Control*, 40:88–121, 1979. 489
8. J. Mycielski and D. Niwiński. Cellular automata on trees, a model for parallel computation. *Fundamenta Informaticae*, XV:139–144, 1991. 490
9. J. Rothstein. On the ultimate limitations of parallel processing. In P. H. Enslow Jr., ed., *Proc. Int. Conf. on Parallel Processing*, p. 206–212, 1976. 488, 489, 494

10. S. Sahni. The DMBC: Architecture and fundamental operations. In *Proc. Int. Conf. on Supercomputing*, p. 60–66, ACM Press, 1995. 488
11. J. L. Trahan, R. Vaidyanathan, R. K. Thiruchelvan. On the power of segmenting and fusing buses. *Journal of Parallel and Distributed Computing*, 34:82–94, 1996. 488, 488
12. J. Wiedermann. Paralelný Turingov stroj — Model distribuovaného počítača. In J. Gruska, ed., *Distribouvané a parallelné systémy*, p. 205–214. Bratislava, 1983. 490

Author Index